Stefano Bistarelli, Andrea Formisano (Eds.)

ICTCS'14

Fifteenth Italian Conference on Theoretical Computer Science

Perugia, Italy, September 17–19, 2014

Proceedings

Editors' address:
Università di Perugia
Dipartimento di Matematica e Informatica
Via Vanvitelli 1
06123 Perugia, Italy

{stefano.bistarelli | andrea.formisano}@unipg.it

Preface

This volume contains the papers presented at ICTCS 2014, the 15th Italian Conference on Theoretical Computer Science held on September 17-19, 2014 in Perugia.

ICTCS is the traditional meeting of the Italian Chapter of the European Association for Theoretical Computer Science (EATCS). The purpose of these meetings is fostering the cross-fertilisation of ideas stemming from different areas of theoretical computer science. Hence, they represent occasions for exchanging ideas and for sharing experiences between researchers. They also provide the ideal environment where junior researchers and PhD students can meet senior researchers.

The Italian Chapter of the EATCS was founded in 1972 and previous meetings took place in Pisa (1972), Mantova (1974 and 1989), L'Aquila (1992), Ravello (1995), Prato (1998), Torino (2001), Bertinoro (2003), Pontignano (2005), Roma (2007), Cremona (2009), Camerino (2010), Varese (2012) and Palermo (2013). As usual, ICTCS 2014 was open to researchers from outside Italy, who are always welcome to submit papers and attend these periodical events.

In this edition, there were 30 submitted contributions. Each of them was reviewed by at least 3 Program Committee members. The Committee decided to accept 26 papers covering several areas of theoretical computer science. The participants came from institutions of various countries, namely, China, Finland, France, India, Israel, Italy, Japan, Poland, Tunisia, Turkey, UK, and USA. The program included two invited speakers, Rocco De Nicola (IMT, Lucca) and Giuseppe Liotta (Università di Perugia) and a presentation given by Flavio Chierichetti (Sapienza Università di Roma), the recipient of the *Young Researcher in Theoretical Computer Science Award 2014*, conferred this year by the Italian Chapter. Furthermore, Livio Bioglio (INSERM, Paris) and Andrea Marino (Università di Milano), the two recipients of the *Best PhD Thesis in Theoretical Computer Science Award 2014*, assigned by the Italian Chapter, gave two talks illustrating their recent research. The program of ICTCS 2014 included a special session devoted to the memory of Alberto Bertoni, which was one of the founders of the Italian Chapter and recently passed away. This session, was organized by Arturo Carpi and Alessandra Cherubini.

We would like to express our gratitude to the invited speakers, to the recipients of the three Awards, and to all authors and participants. We also wish to thank the members of the Program Committee and all additional anonymous reviewers for their hard work. A special mention is due to the colleagues of the Organizing Committee for the invaluable contribution they gave in organizing ICTCS 2014.

We would like to give special thanks to the various sponsors that supported the event: EATCS, Università di Perugia, Dipartimento di Matematica e Informatica, Regione Umbria, Provincia di Perugia, Comune di Perugia, Fondazione Perugiassisi 2019, Fondazione Cassa di Risparmio di Perugia, INdAM-GNCS, IOS Press. Finally, we mention EasyChair and CEUR-WS.org that helped us in organizing the conference and producing the proceedings.

September 2014 — Stefano Bistarelli
Perugia — Andrea Formisano

Program Committee

Paolo Baldan	Università di Padova
Giampaolo Bella	Università di Catania
Marco Bernardo	Università di Urbino
Davide Bilo	Università di Sassari
Stefano Bistarelli (chair)	Università di Perugia
Michele Boreale	Università di Firenze
Tiziana Calamoneri	Sapienza Università di Roma
Antonio Caruso	Università del Salento
Ferdinando Cicalese	Università diSalerno
Flavio Corradini	Università di Camerino
Giorgio Delzanno	Università di Genova
Mariangiola Dezani	Università di Torino
Eugenio Di Sciascio	Politecnico di Bari
Agostino Dovier	Università di Udine
Marco Faella	Università di Napoli “Federico II”
Michele Flammini	Università di L’Aquila
Andrea Formisano (chair)	Università di Perugia
Maurizio Gabbrielli	Università di Bologna
Fabio Gadducci	Università di Pisa
Raffaella Gentilini	Università of Perugia
Laura Giordano	Università del Piemonte Orientale
Giuseppe F. Italiano	Università di Roma ”Tor Vergata”
Sabrina Mantaci	Università di Palermo
Isabella Mastroeni	Università di Verona
Manuela Montangero	Università di Modena e Reggio Emilia
Maurizio Proietti	IASI-CNR, Roma
Antonino Salibra	Università Ca’ Foscari Venezia
Francesco Santini	Università di Perugia
Marinella Sciortino	Università di Palermo
Maurice Ter Beek	ISTI-CNR, Pisa

Local Organizing Committee

Serena Arteritano
Stefano Bistarelli
Arturo Carpi
Andrea Formisano
Raffaella Gentilini
Bruno Iannazzo
Laura Marozzi
Alfredo Navarra
Fernanda Pambianco
Fabio Rossi
Francesco Santini
Simone Topini
Lidia Trotta
Emanuela Ughi
Flavio Vella

Contents

Communications

A formal approach to autonomic systems programming: the SCEL language

Rocco De Nicola

IMT – Institute for Advanced Studies Lucca
rocco.denicola@imtlucca.it

Abstract. The autonomic computing paradigm has been proposed to cope with size, complexity and dynamism of contemporary software-intensive systems. The challenge for language designers is to devise appropriate abstractions and linguistic primitives to deal with the large dimension of systems, and with their need to adapt to the changes of the working environment and to the evolving requirements. We propose a set of programming abstractions that permit to represent behaviours, knowledge and aggregations according to specific policies, and to support programming context-awareness, self-awareness and adaptation. Based on these abstractions, we define SCEL (Software Component Ensemble Language), a kernel language whose solid semantic foundations lay also the basis for formal reasoning on autonomic systems behaviour. To show expressiveness and effectiveness of SCEL's design, we present a Java implementation of the proposed abstractions and show how it can be exploited for programming a robotics scenario that is used as a running example for describing features and potentials of our approach.

Graph drawing beyond planarity: some results and open problems

Giuseppe Liotta

Dipartimento di Ingegneria
Università degli Studi di Perugia, Italy
giuseppe.liotta@unipg.it

Abstract. We briefly review some recent findings and outline some emerging research directions about the theory of "nearly planar" graphs, i.e. graphs that have drawings where some crossing configurations are forbidden.

1 Graph drawing beyond planarity

Recent technological advances have generated torrents of relational data that are hard to display and visually analyze due, mainly, to their large size. Application domains where this need is particularly pressing include Systems Biology, Social Network Analysis, Software Engineering, and Networking. What is required is not simply an incremental improvement to scale up known solutions but, rather, a quantum jump in the sophistication of the visualization systems and techniques. New research scenarios for visual analytics, network visualization, and human-computer interaction paradigms must be identified; new combinatorial models must be defined and their corresponding theoretical problems must be computationally investigated; finally, the theoretical solutions must be experimentally evaluated and put into practice. Therefore, a substantial research effort in the graph drawing and network visualization communities started from the following considerations.

The Planarity Handicap. The classical literature on graph drawing and network visualization showcases elegant algorithms and sophisticated data structures under the assumption that the input relational data set can be displayed as a network where no two edges cross (see, e.g., [14,35,36,40]), i.e. as a planar graph. Unfortunately, almost every graph is non-planar in practice and various experimental studies have established that the human ability of understanding a diagram is dramatically affected by the type and number of edge crossings (see, e.g., [42,43,48]).

Combinatorial Topology vs. Algorithmics. A topological graph is a drawing of a graph in the plane such that vertices are drawn as points and edges are drawn as simple arcs between the points. Extremal theory questions such as "how many edges can a certain type of non-planar topological graph have?" have been investigated by mathematicians for decades, typically under the name of Turán-type problems. However, the corresponding computational question: "How efficiently can one compute a drawing Γ of a non-planar graph such that Γ is a topological graph of a certain type?" has been surprisingly disregarded by the algorithmic community until very recent years.

We recall that planar graphs can be expressed in terms of forbidden subgraphs: A graph G is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$. Then, a fundamental natural step towards understanding non-planar graphs is to consider network visualizations where some types of crossings are forbidden while some other types are allowed. For example, we recall a sequence of HCI experiments by Huang et al. [32,33,34] proving that crossing edges significantly affect human understanding if they form acute angle, while crossing that form angles from about $\frac{\pi}{3}$ to $\frac{\pi}{2}$ guarantee good readability properties. Hence it makes sense to explore complexity issues related to drawings of graphs where such "sharp angle crossings" are forbidden. As another problem, Purchase et al. [42,43,48]) prove that an edge is difficult to read if it is crossed by many other edges; hence, the current research agenda considers computational issues with graph drawings where every edge is crossed by at most k other edges, for a given constant k.

In addition to requiring that some types of edge crossings must be forbidden, non-planar drawings must also satisfy a set of geometric optimization goals (often called *aesthetic requirements*) such as, for example, minimizing the area of the drawing for a given resolution rule, maximizing the aspect ratio, minimizing the number of different slopes used to draw the edges, or the number bends along the edges.

In the next section we briefly recall some of the most recent results in the area and propose a few open problems. More formally, a *drawing* of a graph G: (i) injectively maps each vertex u of G to a point p_u in the plane; (ii) maps each edge (u, v) of G to a Jordan arc connecting p_u and p_v that does not pass through any other vertex; (iii) is such that any two edges have at most one point in common. A drawing of a graph is a *straight-line drawing* if every edge is a straight-line segment, it is a *poly-line drawing* if the edges are polygonal chains and may contain bends.

2 Some results and open problems

The "beyond planarity" research area could be briefly described as the (potentially uncountable) collection of problems of the type depicted in Figure 1, where the column "Forbidden" describes a forbidden crossing configuration and the column "Question" describes a corresponding computational question of interest in graph drawing. We remark that both the forbidden configurations and the computational questions of Figure 1 are mere examples within a much larger research framework. In the remainder, we only give some references about the second and the fourth entry of the table. The interested reader is referred, for example, to recent proceedings of the International Symposium on Graph Drawing [49] for more results on the "beyond planarity" topic. (See also `http://www.graphdrawing.org/symposia.html`.)

2.1 Drawings with large crossing angles

The *crossing angle resolution* of a drawing of a graph measures the smallest angle formed by any pair of crossing edges.

A *RAC drawing* is a drawing of a graph whose edges can cross only orthogonally to one another, i.e. a RAC drawing maximizes the crossing angle resolution. The notion of RAC drawings was first introduced by Didimo et al. in [23], who studied both

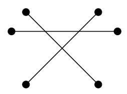
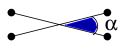
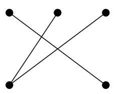
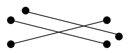
Forbidden configuration	**Algorithmic question**
 no three mutually crossing edges	Complexity of the recognition problem
 no crossing angle smaller than α	Straight−line drawability for graphs with bounded vertex degree
 no fan crossing	Compute simultaneous emebddings of graph pairs
 no edge with k crossings	Compute compact drawings of planar graphs with lmited number of crossings per edge

Fig. 1. A table with some forbidden crossing configurations and related computational questions.

straight-line and poly-line drawings. Variants of RAC drawings are drawings in which the minimum crossing angle must be at least a given constat or the drawings where the minimum crossing angle is exactly a given constant. A limited list of recent papers about RAC drawings and their variants includes [4,5,6,7,15,16,17,18,22,25,47]. See also [24] for more references and open problems about drawing graphs with large crossing angles. A sample open problem follows.

Open Problem: Argyriou *et al.* [6] prove that deciding whether a graph has a straight-line RAC drawing is NP-hard. Hence, maximizing the crossing angle resolution in a straight-line drawing of a graph is also NP-hard. Is there an efficient approximation algorithm for this problem? Is there a polynomial time solution for special families of graphs (e.g. those having bounded vertex degree)?

Related to the problem above, we recall that there is a polynomial time algorithm to recognize whether a bipartite graph has a straight-line RAC drawing such that the vertices of a same partition set all lie on one of two parallel lines [16].

2.2 Drawings with few crossings per edge

For a fixed non negative integer k, a *k-planar drawing* is a drawing of a graph where every edge can be crossed by at most k other edges. A *k-planar graph* is a graph that has a k-planar drawing. Note that the family of 0-planar graphs coincides with the family of planar graphs. The literature about drawings of graphs where every edge can be crossed at most k times has mostly focused on the case $k = 1$.

Concerning Turán-type problems, Pach and Tóth prove that 1-planar graphs with n vertices have at most $4n - 8$ edges, which is a tight upper bound [41]; in the case of straight-line drawings, Didimo [21] proved that a tight bound is $4n - 9$. 1-planarity testing is studied by Korzhik and Mohar who prove that recognizing 1-planar graphs is NP-hard [39]; polynomial-time solutions for the recognition problem are known under some additional assumptions and/or for restricted classes of graphs (see, e.g. [8,27,30]).

Straight-line 1-planar drawings have been studied in [3,31,46]. The relation between 1-planar drawings and RAC drawings is considered in [13,28]. A limited list of additional papers on 1-planar graphs includes [1,2,3,9,10,11,26,29,31,37,38,45].

We conclude with a classical open problem about trade-offs of different aesthetic requirements. Assuming that the vertices are points of an integer grid, the *area of a drawing* of a graph is defined as the area of the smallest axis aligned rectangle that includes the drawing.

Open Problem: It is known that every planar graph with n vertices admits a crossing-free straight-line drawing in $\Theta(n^2)$ area [12,44]. On the other hand, every planar graph can be drawn with straight-line edges in $O(n)$ area if one allows $O(n)$ crossings per edge [50]. Does every planar graph with n vertices have a straight-line drawing with $o(n^2)$ area and a $o(n)$ crossings per edge?.

Starting references to study the above problem include [19,20].

References

1. E. Ackerman. A note on 1-planar graphs. *Discrete Applied Mathematics*, 175:104–108, 2014.
2. E. Ackerman, R. Fulek, and C. D. Tóth. Graphs that admit polyline drawings with few crossing angles. *SIAM J. Discrete Math.*, 26(1):305–320, 2012.
3. M. J. Alam, F. J. Brandenburg, and S. G. Kobourov. Straight-line grid drawings of 3-connected 1-planar graphs. In Wismath and Wolff [49], pages 83–94.
4. P. Angelini, L. Cittadini, W. Didimo, F. Frati, G. D. Battista, M. Kaufmann, and A. Symvonis. On the perspectives opened by right angle crossing drawings. *J. Graph Algorithms Appl.*, 15(1):53–78, 2011.
5. E. N. Argyriou, M. A. Bekos, M. Kaufmann, and A. Symvonis. Geometric rac simultaneous drawings of graphs. *J. Graph Algorithms Appl.*, 17(1):11–34, 2013.
6. E. N. Argyriou, M. A. Bekos, and A. Symvonis. The straight-line rac drawing problem is np-hard. *J. Graph Algorithms Appl.*, 16(2):569–597, 2012.
7. K. Arikushi, R. Fulek, B. Keszegh, F. Moric, and C. D. Tóth. Graphs that admit right angle crossing drawings. *Comput. Geom.*, 45(4):169–177, 2012.
8. C. Auer, C. Bachmaier, F. J. Brandenburg, A. Gleißner, K. Hanauer, D. Neuwirth, and J. Reislhuber. Recognizing outer 1-planar graphs in linear time. In Wismath and Wolff [49], pages 107–118.

9. O. V. Borodin, A. V. Kostochka, A. Raspaud, and E. Sopena. Acyclic colouring of 1-planar graphs. *Discrete Applied Mathematics*, 114(1-3):29–41, 2001.
10. F.-J. Brandenburg, D. Eppstein, A. Gleißner, M. T. Goodrich, K. Hanauer, and J. Reislhuber. On the density of maximal 1-planar graphs. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2012.
11. J. Czap and D. Hudák. 1-planarity of complete multipartite graphs. *Discrete Applied Mathematics*, 160(4-5):505–512, 2012.
12. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
13. H. R. Dehkordi and P. Eades. Every outer-1-plane graph has a right angle crossing drawing. *Int. J. Comput. Geometry Appl.*, 22(6):543–558, 2012.
14. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
15. E. Di Giacomo, W. Didimo, P. Eades, S.-H. Hong, and G. Liotta. Bounds on the crossing resolution of complete geometric graphs. *Discrete Applied Mathematics*, 160(1-2):132–139, 2012.
16. E. Di Giacomo, W. Didimo, P. Eades, and G. Liotta. 2-layer right angle crossing drawings. *Algorithmica*, 68(4):954–997, 2014.
17. E. Di Giacomo, W. Didimo, L. Grilli, G. Liotta, and S. A. Romeo. Heuristics for the maximum 2-layer rac subgraph problem. *The Computer Journal*. on print, published on line on March 2014.
18. E. Di Giacomo, W. Didimo, G. Liotta, and H. Meijer. Area, curve complexity, and crossing resolution of non-planar graph drawings. *Theory Comput. Syst.*, 49(3):565–575, 2011.
19. E. Di Giacomo, W. Didimo, G. Liotta, and F. Montecchiani. h-quasi planar drawings of bounded treewidth graphs in linear area. In M. C. Golumbic, M. Stern, A. Levy, and G. Morgenstern, editors, *WG*, volume 7551 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 2012.
20. E. Di Giacomo, W. Didimo, G. Liotta, and F. Montecchiani. Area requirement of graph drawings with few crossings per edge. *Computational Geometry*, 46(8):909 – 916, 2013.
21. W. Didimo. Density of straight-line 1-planar graph drawings. *Inf. Process. Lett.*, 113(7):236–240, 2013.
22. W. Didimo, P. Eades, and G. Liotta. A characterization of complete bipartite rac graphs. *Inf. Process. Lett.*, 110(16):687–691, 2010.
23. W. Didimo, P. Eades, and G. Liotta. Drawing graphs with right angle crossings. *Theoretical Computer Science*, 412(39):5156–5166, 2011.
24. W. Didimo and G. Liotta. The crossing angle resolution in graph drawing. In J. Pach, editor, *Thirty Essays on Geometric Graph Theory*. Springer, 2012.
25. V. Dujmovic, J. Gudmundsson, P. Morin, and T. Wolle. Notes on large angle crossing graphs. *Chicago J. Theor. Comput. Sci.*, 2011, 2011.
26. P. Eades, S.-H. Hong, N. Katoh, G. Liotta, P. Schweitzer, and Y. Suzuki. Testing maximal 1-planarity of graphs with a rotation system in linear time. In *Proc. of GD 2012*, LNCS. Springer. on print.
27. P. Eades, S.-H. Hong, N. Katoh, G. Liotta, P. Schweitzer, and Y. Suzuki. A linear time algorithm for testing maximal 1-planarity of graphs with a rotation system. *Theor. Comput. Sci.*, 513:65–76, 2013.
28. P. Eades and G. Liotta. Right angle crossing graphs and 1-planarity. *Discrete Applied Mathematics*, 161(7-8):961–969, 2013.
29. I. Fabrici and T. Madaras. The structure of 1-planar graphs. *Discrete Mathematics*, 307(7-8):854–865, 2007.

30. S.-H. Hong, P. Eades, N. Katoh, G. Liotta, P. Schweitzer, and Y. Suzuki. A linear-time algorithm for testing outer-1-planarity. In Wismath and Wolff [49], pages 71–82.
31. S.-H. Hong, P. Eades, G. Liotta, and S.-H. Poon. Fáry's theorem for 1-planar graphs. In J. Gudmundsson, J. Mestre, and T. Viglas, editors, *COCOON*, volume 7434 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2012.
32. W. Huang. Using eye tracking to investigate graph layout effects. In *APVIS*, pages 97–100, 2007.
33. W. Huang, P. Eades, and S.-H. Hong. Larger crossing angles make graphs easier to read. *J. Vis. Lang. Comput.*, 25(4):452–465, 2014.
34. W. Huang, S.-H. Hong, and P. Eades. Effects of crossing angles. In *PacificVis*, pages 41–46, 2008.
35. M. Jünger and P. Mutzel, editors. *Graph Drawing Software*. Springer, 2003.
36. M. Kaufmann and D. Wagner, editors. *Drawing Graphs*. Springer Verlag, 2001.
37. V. P. Korzhik. Minimal non-1-planar graphs. *Discrete Mathematics*, 308(7):1319–1327, 2008.
38. V. P. Korzhik. Proper 1-immersions of graphs triangulating the plane. *Discrete Mathematics*, 313(23):2673–2686, 2013.
39. V. P. Korzhik and B. Mohar. Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory*, 72(1):30–71, 2013.
40. T. Nishizeki and Md.S. Rahman. *Planar Graph Drawing*. World Scientific, 2004.
41. J. Pach and G. Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.
42. H. C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000.
43. H. C. Purchase, D. A. Carrington, and J.-A. Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.
44. W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 138–148, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.
45. Y. Suzuki. Optimal 1-planar graphs which triangulate other surfaces. *Discrete Mathematics*, 310(1):6–11, 2010.
46. C. Thomassen. Rectilinear drawings of graphs. *Journal of Graph Theory*, 12(3):335–341, 1988.
47. M. van Kreveld. The quality ratio of RAC drawings and planar drawings of planar graphs. In *Proc. of GD 2010*, volume 6502 of *LNCS*, pages 371–376. Springer, 2010.
48. C. Ware, H. C. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.
49. S. K. Wismath and A. Wolff, editors. *Graph Drawing - 21st International Symposium, GD 2013, Bordeaux, France, September 23-25, 2013, Revised Selected Papers*, volume 8242 of *Lecture Notes in Computer Science*. Springer, 2013.
50. D. R. Wood. Grid drawings of k-colourable graphs. *Computational Geometry: Theory and Applications*, 30(1):25 – 28, 2005.

Trace complexity

Flavio Chierichetti

Dipartimento di Informatica
Sapienza University of Rome

Abstract. Prediction tasks in machine learning usually require deducing a latent variable, or structure, from observed traces of activity — sometimes, these tasks can be carried out with a significant precision and statistical significance, while sometimes getting any useful prediction requires an unrealistically large number of traces.

In this talk, we will study the trace complexity of (that is, *the number of traces needed for carrying out*) two prediction tasks in social networks: the network inference problem and the number of signers problem.

The first problem [1] consists of reconstructing the edge set of a network given traces representing the chronology of infection times as epidemics spread through the network. The second problem's [2] goal is to guess the unknown number of signers of some email-based petitions, when only a small subset of the emails that circulated is available.

These two examples will allow us to make some general remarks about social networks' prediction tasks.

References

1. B. D. Abrahao, F. Chierichetti, R. Kleinberg, and A. Panconesi. Trace complexity of network inference. In I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman, and R. Uthurusamy, editors, *KDD*, pages 491–499. ACM, 2013.
2. F. Chierichetti, J. M. Kleinberg, and D. Liben-Nowell. Reconstructing patterns of information diffusion from incomplete observations. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *NIPS*, pages 792–800, 2011.

Type disciplines for systems biology

Livio Bioglio

INSERM, UMR-S 1136, iPLESP, Paris, France

Systems Biology is a discipline that aims to study complex biological systems by means of computational models. Because of the complexity of biological behaviors, the formalisms used in this field are usually designed ad-hoc for the biological topic of interest, or they need to be tuned by a long set of evolution rules. Here we present a different approach: we define biological properties through a type discipline, leaving the formalisms as general as possible. We explore three different kinds of Type Systems: a static one, that limits the model that can be written by modelers; a dynamic one, that limits the evolution of the model at run-time; and an hybrid combination of the previous ones.

1 Static type system

Homogeneous biological entities are classified according to their behavior. In order to reproduce such classification, we propose a Minimal Object-Oriented Core Calculus for term-rewriting formalisms. A rewrite system is composed by a term, representing the structure of the modeled system, and a series of reduction rules, representing the possible evolutions of the system: depending on the formalism, these rules can be embedded in terms, like in P Systems [7], or defined in a separate part, like in the Calculus of Looping Sequences [1] (CLS for short). The objective of the OO Core Calculus is to facilitate the organizations of rules, also improving their re-use, and to check the correctness of the model at compile time. In our core calculus it is possible to define classes, that contain methods (encapsulation) and extend another class (subtyping), inheriting all its methods (inheritance). Methods are formed by a sequence of variables, the arguments, and a sequence of reduction rules containing these variables. They are called on symbols of the model, representing biological entities, with a sequence of values as arguments (method invocation). A method invocation is replaced by the reduction rules of the method, in which the variables are replaced by the values used as arguments. These reduction rules are then used for the evolution of the model. The syntax, definitions and rules of the OO Core Calculus are inspired by the ones proposed by Igarashi, Pierce and Wadler for Featherweight Java [6], a minimal core calculus for modeling the Java Type System. For more details, see [2].

2 Dynamic type system

In Biology and Chemistry there can be found several examples of repellency, such as hydrophobicity (the physical property of a molecule that is repelled from a

mass of water), the behavior of anions and cations, or, at a different level of abstraction, the behavior of the rh antigen for the different blood types. As a counterpart, there may be elements, in nature, which always require the presence of other elements: for example, it is difficult to find a lonely atom of oxygen, they always appear in the pair O_2. We bring these aspects at their maximum limit, and, by abstracting away all the phenomena which give rise/arise to/from repellency (and its counterpart), we assume that for each kind of element of our reality we are able to fix a set of elements which are required by the element for its existence, and a set of elements whose presence is forbidden by the element. We enrich the basic CLS with a type discipline which guarantees the soundness of reduction rules with respect to some relevant properties of biological systems deriving from the required and excluded kinds of elements. The key technical tool we use is to associate to each reduction rule the minimal set of conditions an instantiation must satisfy in order to assure that applying this rule to a "correct" system we get a "correct" system as well. We also propose a type inference algorithm, based on the machinery of principal typing [8], and show its soundness and completeness. The required/excluded elements properties modeled here assure, through type inference, that only compatible elements are combined together in the different environments of the biological system took in consideration. Thus the type system intrinsically yields a notion of correct (well-behaving) system according to the expressed requirements. The detailed Type Discipline can be found in [5].
There are cases in which the request/repellency model cannot reflect the behavior of a biological system. An example is homeostasis, the property of a system that regulates its internal environment and tends to maintain stable conditions that are optimal for survival: when this equilibrium is disturbed, built-in regulatory devices respond in order to restore the balance. Different living organisms employ homeostatic mechanisms to maintain some conditions in specific ranges: the human body, like in all the warm-blooded animals, maintain a near-constant body temperature using mechanisms such as vasodilation and vasoconstriction; microorganisms maintain the iron presence above a minimum level to maintain life but up to a maximum level to avoid iron toxicity. For this reason, we propose an extension of the previous Type Disciplines, where we assume that for each element of our system we can fix the minimum and the maximum number of other elements it requires. We enrich CLS with a type discipline and typed reductions that guarantee the soundness of reduction rules with respect to the properties of biological systems deriving from the minimum and the maximum requested numbers of elements, and a type inference algorithm for inferring the type of rewriting rules. Our contribution appeared in [3].

3 Hybrid type system

We present the variant of the Calculus of Looping Sequences with global and local rewrite rules (CLSLR, for short). Global rules are the usual rules of CLS, and they can be applied anywhere in a given term wherever their patterns match the portion of the system under investigation, while local rules can only be applied in the compartment in which they are defined. Terms written in CLSLR are thus syntactically extended to contain explicit local rules within the term, on different compartments. Local rules can be created, moved between different compartments and deleted. Having a calculus in which we can model the dynamic evolution of the rules describing the system allows to study emerging properties of complex systems in a more natural and direct way. As it happens in nature, where data and programs are encoded in the same kind of molecular structures, we insert rewrite rules within the terms modeling the system under investigation. On the other hand, some rule may represent general behaviors, common to the whole system: global rules are used for avoiding the repetition of such rules in each compartment. Since in this framework the focus is put on local rules, we define a set of features that can be associated to each one. Features may define general properties of rewrite rules or properties which are strictly related to the model under investigation. We define a membrane type for the compartments of our model and develop a type systems enforcing the property that a compartment must contain only local rules with specific features. Our framework has been presented in [4].

References

1. R. Barbuti, A. Maggiolo-Schettini, P. Milazzo and A. Troina. A Calculus of Looping Sequences for Modelling Microbiological Systems. *Fundamenta Informaticæ*, 72(1–3):21–35, 2006.
2. L. Bioglio. A Minimal OO Calculus for Modelling Biological Systems. *Computational Models for Cell Processes (CompMod) 2011*, EPTCS 67:50–64, 2011.
3. L. Bioglio. Enumerated Type Semantics for the Calculus of Looping Sequences. *RAIRO - Theoretical Informatics and Applications*, 45(01):35–58, 2011.
4. L. Bioglio, M. Dezani-Ciancaglini, P. Giannini and A. Troina. A Calculus of Looping Sequences with Local Rules. *7th Workshop on Developments in Computational Models (DCM'11)*, EPTCS 88:43–58, 2011.
5. L. Bioglio, M. Dezani-Ciancaglini, P. Giannini and A. Troina. Type Directed Semantics for the Calculus of Looping Sequences. *International Journal of Software and Informatics*, to appear.
6. A. Igarashi, B. Pierce and P. Wadler. Featherweight Java: a Minimal Core Calculus for Java and GJ. *ACM Transactions on Programming Languages and System*, 23:396–450, 2001.
7. G. Păun. *Membrane Computing. An Introduction.* Springer, 2002.
8. J. B. Wells. The Essence of Principal Typings. *International Colloquium on Automata, Languages and Programming (ICALP'02)*, LNCS 2380:913–925, 2002.

Algorithms for biological graphs: analysis and enumeration*

Andrea Marino

Dipartimento di Informatica, Università di Milano, Milano, Italy

The aim of enumeration is to list all the feasible solutions of a given problem satisfying some constraints. Enumeration algorithms are particularly useful whenever the goal of a problem is not clear and all its solutions need to be checked. Since one peculiar property of biological networks is the uncertainty, a scenario in which enumeration algorithms can be helpful is biological network analysis. Modelling biological networks indeed introduce bias: arc dependencies are neglected and underlying hyper-graph behaviours are forced in simple graph representations to avoid intractability. Moreover regulatory interactions between all the biological networks are omitted, even if none of the different biological layers is truly isolated. Last but not least, the dynamical behaviours of biological networks are often not considered: indeed most of the currently available biological network reconstructions are potential networks, where all the possible connections are indicated, even if edges/arcs and vertices are hardly present all together at the same time. More details about these aspects of the biological networks can be found in [8].

Our Contribution. We have shown four examples of enumeration algorithms that can be applied to efficiently deal with some biological problems modelled by using biological networks: enumerating central and peripheral nodes of a network, enumerating stories, enumerating paths or cycles, and enumerating bubbles. Notice that the corresponding computational problems we define are of more general interest and our results hold in the case of arbitrary networks.

1 Enumerating central and peripheral vertices

Structural analysis allows the identification of important and not important vertices within a network and also for this reason has become very popular in many disciplines. In the biological domain, the importance of a vertex can be defined in many different ways. With neighbourhood-based centrality measures, such as degree, the importance of the vertices is inferred from their local connectivity and the more connections a vertex has the more central it is. Closeness, eccentricity, and shortest path based betweenness relies on global properties of a network, such as distance between vertices.

* The author wants to thank the PhD advisor Pierluigi Crescenzi, the PhD School of Dipartimento di Sistemi e Informatica, Università di Firenze (Italy), and all the coauthors of the papers.

We have focused on the enumeration of the radial and diametral vertices, i.e. vertices that are central and peripheral according to the eccentricity notion of centrality, and on the computation of the radius and diameter of biological networks and of real world graphs in general. The diameter and radius of a graph are respectively the maximum and minimum eccentricity among all its nodes, where the eccentricity of a node x is the distance from x to its farthest node. Thus, intuitively, the diametral source vertices are the vertices that hardly reach the other ones, the diametral target vertices are the vertices hardly reachable from the other ones, and the radial vertices are the vertices that easily reach all the vertices of the network. In order to calculate the vertices that can be easily reached from any other vertex, it is sufficient to consider the transposed graph.

We have presented the D*i*FUB Algorithm, which is able to list all the diametral sources and targets and to compute the diameter of (strongly) connected components of a graph $G = (V, E)$ in time $O(|E|)$ in practice, even if, in the worst case, the complexity is $\Theta(|V||E|)$. Analogously, we have presented a new algorithm to list all the central vertices and to compute the radius of (strongly) connected components of a graph in *almost* $O(|E|)$ time in practice.

The analysis of real world networks in general, such as citation, collaboration, communication, road, social, and web networks, has attracted a lot of attention. The fundamental analysis measures have been reviewed in [12]. Moreover the size of these networks has been increasing rapidly, so that in order to study such measures, algorithms able to handle huge amount of data are needed. Since the algorithms available until now were not able to compute diameter and radius in the case of huge real world graphs, the contribution of our algorithms is not just limited to biological networks analysis, but extends also to the analysis of complex networks in general. We thus have shown their effectiveness also for several other kinds of complex networks. More details can be found in the work [5], which has been the generalization of [4, 3]. Our algorithm in [3] has been used to compute the diameter of Facebook Network (721.1M vertices, 68.7G edges, and diameter 41) with just 17 BFSes in a popular work ([9], divulged by New York Times on November 22, 2011).

2 Enumerating stories

The problem of enumerating stories was motivated initially by the biological question in [10] related to Metabolic networks, in particular to compound graphs, in which vertices are compounds and there is an arc from a compound x to a compound y if there is a metabolic reaction that consumes x and produces y. A subset $\mathbb{B}$ corresponds to compounds that have been experimentally identified as having a significantly higher or lower production in a given condition (for instance when an organism is exposed to some stress). The aim is then to extract all the interaction dependencies among the compounds in $\mathbb{B}$ which do not create cycles but at the same time involve as many compounds as possible. These may require intermediate steps that concern compounds not in $\mathbb{B}$, but the initial and final steps must involve only compounds in $\mathbb{B}$. A solution, that is a possible scenario of metabolic dependencies, is called a *(metabolic) story*.

A metabolic story has to capture the relationship between the vertices of interest in a way that allows us to define a flow of matter from a set of sources to a set of target compounds. The need for this hierarchy between the compounds led us to consider acyclic solutions. The maximality condition has been added in order to capture all alternative paths between the sources and the targets. The problem is then to "tell" all possible stories given as input a graph G and a subset $\mathbb{B}$ of the vertices of G.

We have presented a polynomial algorithm to find one story and an exact but exponential approach for the enumeration problem [1]. This definition is a generalization of a well-known problem which is the *feedback arc set* problem. However, any polynomial-delay algorithm to enumerate feedback arc sets (ex: [14]) can only be used in some particular instances. Moreover we have shown that finding a story with a specified set of sources or targets is NP-hard.

Our contribution appeared in [1] and its biological application in [11].

3 Enumerating cycles or paths

Studying paths or cycles of biological networks can be useful for several purposes. In the case of interaction graphs, such as Gene Regulatory networks, the importance of enumeration has been shown in [7]. These networks are directed, their vertices are genes, and their arcs are signed, where the sign or weight of the arcs indicates the causal relationship between the vertices, such as activation or inhibition. In particular cycles and paths can be useful for studying dependencies among vertices, the steady state and multistationarity of dynamic models.

We have considered the problem of enumerating paths and cycles in the case of undirected graphs. This result can be useful for undirected Protein-Protein Interaction networks, where nodes are proteins and edges are interactions, but in the case of interaction networks in general, our approach neglects the effects of the controls, i.e. the sign and direction of the arcs. In this latter case, the cycles can be enumerated in the underlying undirected graph and *a posteriori* filtered or *ad hoc* algorithms can be applied. The main question arising from our work, is whether it is possible to extend our result to directed graphs in order to efficiently deal also with this kind of networks.

On the other hand, our contribution is not just restricted to biological undirected networks, but extends also to arbitrary undirected graphs. Listing all the paths and cycles in a graph is a classical problem whose efficient solutions date back to the early 70s. The best known solution in the literature is given by Johnson's algorithm [6] and takes $O((|\mathcal{C}(G)| + 1)(|E| + |V|))$ and $O((|\mathcal{P}_{st}(G)| + 1)(|E| + |V|))$ time for a graph $G = (V, E)$, where $\mathcal{C}(G)$ and $\mathcal{P}_{st}(G)$ denote respectively the set of cycles and (s, t)-paths in G. However there exists graphs for which this algorithm is not optimal.

We have presented the first optimal algorithm to list all the paths and cycles in an undirected graph G. Our algorithm requires $O(|E| + \sum_{c \in \mathcal{C}(G)} |c|)$ time and is asymptotically optimal: indeed, $\Omega(|E|)$ time is necessarily required to read G as input, and $\Omega(\sum_{c \in \mathcal{C}(G)} |c|)$ time is necessarily required to list the output. Moreover, our algorithm lists all the (s,t)-paths in G optimally in $O(|E| + \sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$ time, observing that $\Omega(\sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$ time is necessarily required to list the output.

Our algorithm exploits the decomposition of the graph into biconnected components and without loss of generality restricts to study paths and cycles in a same biconnected component. Thus it recursively lists the cycles or (s,t)-paths using the classical binary partition: given an edge e in G, list all the solutions containing e, and then all the solutions not containing e, at each time modifying the graph. In order to avoid recursive calls (in the binary partition) that do not list solutions, we have used a *certificate*, as a data structure, whose cost for dynamically updating is constant with respect to the number of solutions produced. In order to prove the complexity obtained, we have exploited the properties of the binary recursion tree corresponding to the binary partition. For more details, see [2].

4 Enumerating bubbles

A DNA fragment, that is an RNA-coding sequence, is transformed in a Pre-mRNA sequence, through the transcription phase, in which sequences of *exons* and sequences of *introns* alternatively occur. The removal of all the sequences of introns and of some sequences of exons leads to the mRNA sequence, that is a protein-coding sequence, that translated leads to a protein. Since not any exon is transcribed in the mRNA sequence, there can be many possible mRNA sequences. For instance, let $\langle e_1, i_1, e_2, i_2, e_3, i_3, e_4, i_4 \rangle$ be a fragment of DNA, where for any j, with $1 \leq j \leq 3$, e_j and i_j are the j-th sequence of exons and introns respectively. The possible resulting mRNA sequences containing e_1 are $\langle e_1, e_2, e_3, e_4 \rangle$, $\langle e_1, e_2, e_3 \rangle$, $\langle e_1, e_2, e_4 \rangle$, $\langle e_1, e_3, e_4 \rangle$, $\langle e_1, e_2 \rangle$, $\langle e_1, e_3 \rangle$, $\langle e_1, e_4 \rangle$. The underlying phenomenon is called alternative splicing and checking all the alternative events has been shown in [13] to correspond to checking recognisable patterns in a de Bruijn graph built from the reads provided by a sequencing project. The pattern corresponds to an (s,t)-bubble: an (s,t)-bubble is a pair of vertex-disjoint (s,t)-paths that only shares s and t.

Since the k-mers correspond to all words of length k present in the reads (strings) of the input dataset, and only those, in relation to the classical de Bruijn graph for all possible words of size k, the de Bruijn graph for NGS data may then not be complete. We have ignored all the details related to the treatment of NGS data using De Bruijn graphs, and consider instead the more general case of finding all (s,t)-bubbles in an arbitrary directed graph. In particular we show the first linear delay algorithm to identify all bubbles. A previous known algorithm presented in [13] was an adaptation of Tiernan's algorithm for cycle enumeration [15] which does not have a polynomial delay. In the worst case the time elapsed between the output of two solutions is proportional to the number of paths in the graph, i.e. exponential in the size of the graph. Our algorithm is a non trivial adaptation of Johnson's cycle enumeration algorithm [6] in a directed graph with the same theoretical complexity. Notably, the method we propose enumerates all bubbles with a given source with $O(|V|+|E|)$ delay. The algorithm requires an initial transformation of the graph, for each source s, that takes $O(|V|+|E|)$ time and space; this transformation reduces the enumeration of bubbles to the enumeration of constrained cycles in a special graph.

References

1. V. Acuña, E. Birmelé, L. Cottret, P. Crescenzi, F. Jourdan, V. Lacroix, A. Marchetti-Spaccamela, A. Marino, P. V. Milreu, M.-F. Sagot, and L. Stougie. Telling stories: Enumerating maximal directed acyclic graphs with a constrained set of sources and targets. *Theor. Comput. Sci.*, 457:1–9, 2012.
2. E. Birmelé, R. A. Ferreira, R. Grossi, A. Marino, N. Pisanti, R. Rizzi, and G. Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In *SODA*, pages 1884–1896, 2013.
3. P. Crescenzi, R. Grossi, M. Habib, L. Lanzi, and A. Marino. On computing the diameter of real-world undirected graphs. *Theor. Comput. Sci.*, 514:84–95, 2013.
4. P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, and A. Marino. Finding the diameter in real-world graphs - experimentally turning a lower bound into an upper bound. In *ESA (1)*, pages 302–313, 2010.
5. P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino. On computing the diameter of real-world directed (weighted) graphs. In *SEA*, pages 99–110, 2012.
6. D.B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84, 1975.
7. S. Klamt and A. von Kamp. Computing paths and cycles in biological interaction graphs. *BMC Bioinformatics*, 10:181, 2009.
8. C. Klein, A. Marino, M.-F. Sagot, P.V. Milreu, and M. Brilli. Structural and dynamical analysis of biological networks. *Briefings in functional genomics*, 2012.
9. B. Lars, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four degrees of separation. In *WebSci*, pages 33–42, 2012.
10. G. Madalinski, E. Godat, S. Alves, D. Lesage, E. Genin, P. Levi, J. Labarre, J.-C. Tabet, E. Ezan, and C. Junot. Direct introduction of biological samples into a ltq-orbitrap hybrid mass spectrometer as a tool for fast metabolome analysis. *Analytical Chemistry*, 80(9):3291–3303, 2008.
11. P. V. Milreu, C. Klein, L. Cottret, V. Acuña, E. Birmelé, M. Borassi, C. Junot, A. Marchetti-Spaccamela, A. Marino, L. Stougie, F. Jourdan, P. Crescenzi, V. Lacroix, and M.-F. Sagot. Telling metabolic stories to explore metabolomics data: a case study on the yeast response to cadmium exposure. *Bioinformatics*, 30(1):61–70, 2014.
12. M. E. J. Newman. The structure and function of complex networks. *SIAM REVIEW*, 45:167–256, 2003.
13. G. Sacomoto, J. Kielbassa, R. Chikhi, R. Uricaru, P. Antoniou, M.-F. Sagot, P. Peterlongo, and V. Lacroix. Kissplice: de-novo calling alternative splicing events from rna-seq data. *BMC Bioinformatics*, 13(S-6):S5, 2012.
14. B. Schwikowski and E. Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1-3):253 – 265, 2002.
15. J. C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Communonications ACM*, 13:722–726, 1970.

Timed process calculi: from durationless actions to durational ones [⋆]

Marco Bernardo[1] Flavio Corradini[2] Luca Tesei[2]

[1] Dipartimento di Scienze di Base e Fondamenti, Università di Urbino, Italy
[2] Scuola di Scienze e Tecnologie, Università di Camerino, Italy

Abstract. Several timed process calculi have been proposed in the literature, which mainly differ for the way in which delays are represented. In particular, a distinction is made between integrated-time calculi, in which actions are durational, and orthogonal-time calculi, in which actions are instantaneous and delays are expressed separately. To reconcile the two approaches, in a previous work an encoding from the integrated-time calculus CIPA to the orthogonal-time calculus TCCS was defined, which preserves timed bisimilarity. To complete the picture, in this paper we consider the reverse translation, by examining the modifications to the two calculi that are needed to make an encoding feasible, as well as the behavioral equivalence that is appropriate to preserve. We then introduce an encoding from modified TCCS to modified CIPA, and show that it can only preserve the weak variant of timed bisimilarity.

1 Introduction

Computing systems are characterized not only by their functional behavior, but also by their quantitative features. In particular, *timing aspects* play a fundamental role, as they describe the temporal evolution of system activities. This is especially true for *real-time systems*, which are considered correct only if the execution of their activities fulfills certain *temporal constraints*.

When modeling these systems, time is represented through nonnegative numbers. In the following, we refer to *abstract time*, in the sense that we use time as a parameter for expressing constraints about instants of occurrences of actions. Unlike *physical time*, abstract time permits simplifications that are convenient, on the conceptual side, to obtain tractable models.

Many *timed process calculi* have appeared in the literature. Among them, we mention temporal CCS [8], timed CCS [15], timed CSP [13], real-time ACP [2], urgent LOTOS [4], CIPA [1], TPL [7], ATP [11], TIC [12], and PAFAS [6]. As observed in [10, 14, 5], these calculi differ on the basis of a number of time-related options, some of which are recalled below:

– *Durationless actions* versus *durational actions*. In the first case, actions are instantaneous events and time passes in between them; hence, functional

[⋆] Work partially supported by the MIUR-PRIN project CINA.

behavior and time are *orthogonal*. In the second case, every action takes a fixed amount of time to be performed and time passes only due to action execution; hence, functional behavior and time are *integrated*.
– *Relative time* versus *absolute time*. Assume that timestamps are associated with the events observed during system execution. In the first case, each timestamp refers to the time instant of the previous observation. In the second case, all timestamps refer to the starting time of the system execution.
– *Global clock* versus *local clocks*. In the first case, there is a single clock that governs time passing. In the second case, there are several clocks associated with the various system parts, which elapse independent of each other although they define a unique notion of global time.

Moreover, for timed process calculi, there are several different interpretations of action execution, in terms of whether and when it can be delayed, such as:

– *Eagerness*: actions must be performed as soon as they become enabled, i.e., without any delay, thereby implying that they are urgent.
– *Laziness*: after getting enabled, actions can be delayed arbitrarily long before they are executed.
– *Maximal progress*: enabled actions can be delayed arbitrarily long unless they are involved in synchronizations, in which case they are urgent.

In this paper, we focus on two different timed process calculi obtained by suitably combining the time-related options mentioned above. More precisely, the first calculus, TCCS [8], is inspired by the *two-phase functioning principle*, according to which actions are durationless, time is relative, and there is a single global clock. In contrast, the second calculus, CIPA [1], is inspired by the *one-phase functioning principle*, according to which actions are durational, time is absolute, and several local clocks are present.

In [5], it was shown that some of the choices concerned with the time-related options and action execution interpretations are not irreconcilable, thus permitting the interchange of concepts and analysis techniques. More precisely, the different expressive power of the two considered process calculi was investigated by developing a bisimulation-semantics-preserving encoding of CIPA processes into TCCS processes for each action execution interpretation.

In this paper, we complete the previous expressiveness study by considering the reverse encoding from TCCS processes to CIPA processes, which may also be exploited for checking bisimilarity of TCCS processes more efficiently. As pointed out at the end of [5], there are several issues that need to be addressed before the reverse encoding can be established. Our first contribution is to provide a solution for each of the various problems. Our second contribution is the definition of the reverse encoding, together with a full abstraction result of this reverse encoding under *weak* timed bisimilarity, as opposed to the direct encoding demonstrated to be fully abstract with respect to *strong* timed bisimilarity in [5].

The rest of the paper is organized as follows. In Sect. 2, we recall TCCS and CIPA. In Sect. 3, we discuss the main design decisions behind the reverse encoding. In Sect. 4, we define the reverse encoding and show that it preserves weak timed bisimilarity. Finally, in Sect. 5 we provide some concluding remarks.

2 Background

2.1 Preliminaries

We denote by A a nonempty set of visible actions – ranged over by a, b – and by $\bar{A} = \{\bar{a} \mid a \in A\}$ the set of corresponding coactions such that $\bar{\bar{a}} = a$ for all $a \in A$. We use $Act = A \cup \bar{A} \cup \{\tau\}$ to indicate the set of all actions – ranged over by α, β – where τ is the invisible action.

We denote by Rel a set of action relabeling functions. Each such function $\varphi : Act \to Act$ satisfies $\varphi(\tau) = \tau$ and $\overline{\varphi(a)} = \varphi(\bar{a})$ for all $a \in Act \setminus \{\tau\}$.

We denote by $\mathcal{T} = (T, \boxplus, \sqsubseteq)$ a time domain such that $T \cap Act = \emptyset$, which is equipped with an associative operation $\boxplus$ possessing neutral element and a total order relation $\sqsubseteq$ satisfying $t_1 \sqsubseteq t_2$ iff there exists $t' \in T$ such that $t_1 \boxplus t' = t_2$. Typical choices are $T = \mathbb{N}$ and $T = \mathbb{R}_{\geq 0}$, with the usual $+$ and $\leq$.

Finally, we denote by Var a nonempty set of process variables – ranged over by X, Y – whose occurrences can be free or bound by "rec".

2.2 Durationless Actions: TCCS

We recall from [8] the syntax of TCCS. As in [5], we leave out the idling operator δ and the weak choice operator $\oplus$, as they have no direct counterpart in CIPA.

Definition 1. *The set of process terms of the process language $\mathcal{PL}_{\text{TCCS}}$ is generated by the following syntax:*

$P ::=$	$\mathbf{0}$	*stopped process*
$\mid$	$\alpha.P$	*action prefix*
$\mid$	$(t).P$	*delay prefix*
$\mid$	$P + P$	*alternative composition*
$\mid$	$P \mid P$	*parallel composition*
$\mid$	$P \backslash L$	*restriction*
$\mid$	$P[\varphi]$	*relabeling*
$\mid$	X	*process variable*
$\mid$	$\text{rec}\, X : P$	*recursion*

where $\alpha \in Act$, $t \in \mathbb{N}_{>0}$, $L \subseteq A$, $\varphi \in Rel$, and $X \in Var$. We denote by $\mathbb{P}_{\text{TCCS}}$ the set of closed and guarded process terms of $\mathcal{PL}_{\text{TCCS}}$. ■

Process $\mathbf{0}$ can neither proceed with any action, nor proceed through time. Process $\alpha.P$ can perform instantaneous action α and then evolves into process P; action α is urgent, hence time cannot progress before α is executed. Process $(t).P$ evolves into process P after a delay equal to t.

Process $P_1 + P_2$ represents a nondeterministic choice between processes P_1 and P_2, with the choice being resolved depending on whether an action of P_1 or P_2 is executed first. Time does not resolve choices, in the sense that any initial passage of time common to P_1 and P_2 must be allowed without making the choice. Process $P_1 | P_2$ describes the parallel composition of processes P_1 and P_2,

$\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$	$\dfrac{}{(t).P \overset{t}{\rightsquigarrow} P}$
$\dfrac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'}$ $\quad$ $\dfrac{P_2 \xrightarrow{\alpha} P_2'}{P_1 + P_2 \xrightarrow{\alpha} P_2'}$	$\dfrac{}{(t+t').P \overset{t}{\rightsquigarrow} (t').P}$ $\quad$ $\dfrac{P \overset{t}{\rightsquigarrow} P'}{(t').P \overset{t+t'}{\rightsquigarrow} P'}$
$\dfrac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2}$ $\quad$ $\dfrac{P_2 \xrightarrow{\alpha} P_2'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1 \mid P_2'}$	$\dfrac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_2 \overset{t}{\rightsquigarrow} P_2'}{P_1 + P_2 \overset{t}{\rightsquigarrow} P_1' + P_2'}$
$\dfrac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{\bar{a}} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$	$\dfrac{P_1 \overset{t}{\rightsquigarrow} P_1' \quad P_2 \overset{t}{\rightsquigarrow} P_2'}{P_1 \mid P_2 \overset{t}{\rightsquigarrow} P_1' \mid P_2'}$
$\dfrac{P \xrightarrow{\alpha} P' \quad \alpha \notin L \cup \bar{L}}{P \backslash L \xrightarrow{\alpha} P' \backslash L}$	$\dfrac{P \overset{t}{\rightsquigarrow} P'}{P \backslash L \overset{t}{\rightsquigarrow} P' \backslash L}$
$\dfrac{P \xrightarrow{\alpha} P'}{P[\varphi] \xrightarrow{\varphi(\alpha)} P'[\varphi]}$	$\dfrac{P \overset{t}{\rightsquigarrow} P'}{P[\varphi] \overset{t}{\rightsquigarrow} P'[\varphi]}$
$\dfrac{P\{\mathrm{rec}\, X : P \hookrightarrow X\} \xrightarrow{\alpha} P'}{\mathrm{rec}\, X : P \xrightarrow{\alpha} P'}$	$\dfrac{P\{\mathrm{rec}\, X : P \hookrightarrow X\} \overset{t}{\rightsquigarrow} P'}{\mathrm{rec}\, X : P \overset{t}{\rightsquigarrow} P'}$

Table 1. Structural operational semantic rules for TCCS

where any two complementary actions may synchronize thereby resulting in a τ action; also in this case, any initial passage of time must be permitted.

Process $P \backslash L$ behaves as process P except for actions in $L \cup \bar{L}$, which are forbidden; this operator is useful to force synchronizations between complementary actions. Process $P[\varphi]$ behaves as process P, with the difference that every performed action is transformed via φ; this operator allows processes with different actions to communicate. Finally, $\mathrm{rec}\, X : P$ represents a recursive process, which behaves as process P in which every free occurrence of X is replaced by $\mathrm{rec}\, X : P$ itself; the resulting process will be denoted by $P\{\mathrm{rec}\, X : P \hookrightarrow X\}$.

Following [8], the intuitive meaning of process terms is formalized in Table 1. Transition relation $\longrightarrow$ on the left represents the functional behavior. Transition relation $\rightsquigarrow$ on the right represents the timing behavior according to time additivity (second and third rules) and time determinism (fourth and fifth rules); the second rule is necessary for the applicability of the fourth and fifth ones, while the third rule is necessary for the forthcoming equivalence.

A notion of weak bisimilarity for TCCS was studied in [9]. It is an extension of Milner's weak bisimilarity that is capable of summing up delays while abstracting from τ actions. Weak transitions are defined as follows:

- $\Longrightarrow = (\xrightarrow{\tau})^*$.
- $\overset{a}{\Longrightarrow} = \Longrightarrow\xrightarrow{a}\Longrightarrow$.
- $\overset{\hat{\alpha}}{\Longrightarrow} = \Longrightarrow$ if $\alpha = \tau$, $\overset{\hat{\alpha}}{\Longrightarrow} = \overset{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$.
- $\overset{t}{\Longrightarrow} = \Longrightarrow\overset{t_1}{\leadsto}\Longrightarrow \cdots \Longrightarrow\overset{t_n}{\leadsto}\Longrightarrow$ where $t = \sum_{1 \leq i \leq n} t_i$, $n \in \mathbb{N}_{\geq 1}$.

Definition 2. *A symmetric relation $\mathcal{B}$ over $\mathbb{P}_{\mathrm{TCCS}}$ is a* weak timed bisimulation *iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all actions $\alpha \in Act$ and delays $t \in \mathbb{N}_{>0}$:*

- *For each $P_1 \xrightarrow{\alpha} P_1'$ there exists $P_2 \overset{\hat{\alpha}}{\Longrightarrow} P_2'$ such that $(P_1', P_2') \in \mathcal{B}$.*
- *For each $P_1 \overset{t}{\leadsto} P_1'$ there exists $P_2 \overset{t}{\Longrightarrow} P_2'$ such that $(P_1', P_2') \in \mathcal{B}$.*

$P_1 \approx_{\mathrm{TCCS}} P_2$ iff (P_1, P_2) is contained in a weak timed bisimulation. ■

2.3 Durational Actions: CIPA

We recall from [1] the syntax of CIPA. As in [5], we add the relabeling operator.

Definition 3. *The set of process terms of the process language $\mathcal{PL}_{\mathrm{CIPA}}$ is generated by the following syntax:*

Q	$::=$	nil	*inactive process*
	$\mid$	$a.Q$	*durational action prefix*
	$\mid$	wait $t.Q$	*waiting prefix*
	$\mid$	$Q + Q$	*alternative composition*
	$\mid$	$Q \mid Q$	*parallel composition*
	$\mid$	$Q \backslash L$	*restriction*
	$\mid$	$Q[\varphi]$	*relabeling*
	$\mid$	X	*process variable*
	$\mid$	rec $X : Q$	*recursion*

where $a \in Act \setminus \{\tau\}$, $t \in \mathbb{N}_{>0}$, $L \subseteq A$, $\varphi \in Rel$, and $X \in Var$. We denote by $\mathbb{P}_{\mathrm{CIPA}}$ the set of closed and guarded process terms of $\mathcal{PL}_{\mathrm{CIPA}}$. ■

Process nil cannot proceed with any action, but can let time pass. Process $a.Q$ can perform urgent action a and evolves into process Q after the execution of a has finished; all occurrences of an action are assumed to have the same duration, which is established by a function $\Delta : (Act \backslash \{\tau\}) \to \mathbb{N}_{>0}$ such that $\Delta(\bar{a}) = \Delta(a)$. Process wait $t.Q$ waits for time t and then becomes process Q. All the other operators work as expected, with the additional constraints that each relabeling function φ must preserve durations, i.e., $\Delta(\varphi(a)) = \Delta(a)$ for all $a \in Act \setminus \{\tau\}$, and any pair of actions a and $\bar{a}$ can synchronize only if they start at the same time, yielding a τ action with the same duration as the two original actions.

Following [1], the set $\mathbb{KP}$ of states correspond to process terms augmented with local clocks, so to keep track of the time elapsed in the various sequential components. The shorthand $t \Rightarrow Q$ means that the clock value $t \in \mathbb{N}_{\geq 0}$ is distributed over all subprocesses of Q according to the extended syntax for $\mathbb{KP}$:

$$\frac{}{t \Rightarrow a.Q \xrightarrow[\Delta(a)]{a@t} (t+\Delta(a)) \Rightarrow Q} \qquad \frac{}{t \Rightarrow \mathrm{wait}\, t'.Q \xrightarrow[t']{\tau@t} (t+t') \Rightarrow Q}$$

$$\frac{K_1 \xrightarrow[d]{\alpha@t} K_1' \quad \neg(K_2 \xrightarrow[d']{\alpha'@t'} K_2' \,\wedge\, t' < t)}{K_1 + K_2 \xrightarrow[d]{\alpha@t} K_1'} \qquad \frac{K_2 \xrightarrow[d]{\alpha@t} K_2' \quad \neg(K_1 \xrightarrow[d']{\alpha'@t'} K_1' \,\wedge\, t' < t)}{K_1 + K_2 \xrightarrow[d]{\alpha@t} K_2'}$$

$$\frac{K_1 \xrightarrow[d]{\alpha@t} K_1' \quad \neg(K_2 \xrightarrow[d']{\alpha'@t'} K_2' \,\wedge\, t' < t)}{K_1|K_2 \xrightarrow[d]{\alpha@t} K_1'|K_2} \qquad \frac{K_2 \xrightarrow[d]{\alpha@t} K_2' \quad \neg(K_1 \xrightarrow[d']{\alpha'@t'} K_1' \,\wedge\, t' < t)}{K_1|K_2 \xrightarrow[d]{\alpha@t} K_1|K_2'}$$

$$\frac{K_1 \xrightarrow[d]{a@t} K_1' \quad K_2 \xrightarrow[d]{\bar{a}@t} K_2'}{K_1|K_2 \xrightarrow[d]{\tau@t} K_1'|K_2'} \qquad \frac{K \xrightarrow[d]{\alpha@t} K' \quad \alpha \notin L \cup \bar{L}}{K\backslash L \xrightarrow[d]{\alpha@t} K'\backslash L}$$

$$\frac{K \xrightarrow[d]{\alpha@t} K'}{K[\varphi] \xrightarrow[d]{\varphi(\alpha)@t} K'[\varphi]} \qquad \frac{t \Rightarrow Q\{\mathrm{rec}\, X : Q \hookrightarrow X\} \xrightarrow[d]{\alpha@t} K'}{t \Rightarrow \mathrm{rec}\, X : Q \xrightarrow[d]{\alpha@t} K'}$$

Table 2. Structural operational semantic rules for CIPA

$$\begin{aligned} K \ ::= \ & t \Rightarrow \mathrm{nil} \mid t \Rightarrow a.Q \mid t \Rightarrow \mathrm{wait}\, t'.Q \mid t \Rightarrow \mathrm{rec}\, X : Q \mid \\ & K + K \mid K|K \mid K\backslash L \mid K[\varphi] \end{aligned}$$

In this setting, any transition is of the form $K \xrightarrow[d]{\alpha@t} K'$, meaning that $K \in \mathbb{KP}$ performs an action of name $\alpha \in \mathit{Act}$ that starts at time $t \in \mathbb{N}_{\geq 0}$ and has duration $d \in \mathbb{N}_{>0}$, after which evolves to $K' \in \mathbb{KP}$. The transition relation is defined in Table 2, where negative premises are present as in [5]. Those in the rules for alternative composition enforce action urgency. Those in the rules for parallel composition avoid the generation of *ill-timed paths*, i.e., computations along which the starting time of some actions decreases as the execution proceeds.

A notion of weak bisimilarity for CIPA was studied in [1] under the name of timed branching bisimilarity, which has the capability of summing up consecutive waitings. Weak transitions are defined as follows: $\Longrightarrow = \xrightarrow[d_1]{\tau@t_1} \cdots \xrightarrow[d_n]{\tau@t_n}$, $n \in \mathbb{N}$.

Definition 4. *A symmetric relation $\mathcal{B}$ over $\mathbb{KP}$ is a* weak timed bisimulation *iff, whenever $(K_1, K_2) \in \mathcal{B}$, then for all actions $\alpha \in Act$, starting times $t \in \mathbb{N}_{\geq 0}$, and durations $d \in \mathbb{N}_{>0}$ it holds that for each $K_1 \xrightarrow[d]{\alpha@t} K_1'$:*

– *When $\alpha \neq \tau$, there exists $K_2 \Longrightarrow K_2'' \xrightarrow[d]{\alpha@t} K_2'$ such that $(K_1, K_2'') \in \mathcal{B}$ and $(K_1', K_2') \in \mathcal{B}$.*

– *When* $\alpha = \tau$*, either* $(K'_1, K_2) \in \mathcal{B}$*, or there exists* $K_2 \Longrightarrow K''_2 \xrightarrow[d']{\alpha @ t'} K'_2$ *such that* $(K_1, K''_2) \in \mathcal{B}$ *and* $(K'_1, K'_2) \in \mathcal{B}$*.*

$K_1 \approx_{\text{CIPA}} K_2$ *iff* (K_1, K_2) *is contained in a weak timed bisimulation. Moreover,* $Q_1 \approx_{\text{CIPA}} Q_2$ *iff* $(0 \Rightarrow Q_1, 0 \Rightarrow Q_2)$ *is contained in a weak timed bisimulation.* ■

Although in the clause $\alpha = \tau$ it may be $t' \neq t$ and $d' \neq d$, possible subsequent visible actions must start at the same time in both processes for $\approx_{\text{CIPA}}$ to hold.

3 Design of the Reverse Encoding

In [5], where an encoding from CIPA to TCCS was proposed, a number of issues were raised about the existence of a reverse encoding from TCCS back to CIPA. In this section, we recall those issues and discuss how to address them.

3.1 Adapting TCCS and CIPA

The first issue is related to the range of values that can be used in CIPA to express action durations. In TCCS, it is possible to describe both timed processes and untimed ones. Consider for example the untimed TCCS process $a.b.\mathbf{0}$. This cannot be translated into a reasonably corresponding CIPA process for the very simple reason that actions a and b are instantaneous, but CIPA does not allow zero durations. Moreover, due to instantaneous actions, TCCS processes may exhibit Zeno behaviors, which are not possible in CIPA. For instance, the timed TCCS process $(t_1).a.\text{rec}\,X : (b.X + c.(t_2).\mathbf{0})$ may perform, after time t_1 and action a, an arbitrary (even infinite) number of actions b at the same time. These problems can be straightforwardly solved by admitting zero durations in CIPA through an extended duration function $\Delta : (Act \setminus \{\tau\}) \to \mathbb{N}$.

The second issue that we address is timelock. In a TCCS process, time does not solve choices; indeed, the operational rules for alternative and parallel composition allow time to pass only if all the subprocesses do so. As a consequence, a local timelock always implies a global timelock, which may in turn determine a deadlock. By contrast, in CIPA timelock cannot occur unless there is a deadlock, because time passing is associated with action execution and explicit waiting. Consider the TCCS process $\mathbf{0} + (t).\mathbf{0}$ and the ideally corresponding CIPA process $\text{nil} + \text{wait}\,t.\text{nil}$; the former process cannot let time pass, while the latter process can. The same would happen with $(a.\mathbf{0})\backslash\{a\} + (t).\mathbf{0}$ and $(a.\text{nil})\backslash\{a\} + \text{wait}\,t.\text{nil}$.

To avoid timelocks due to the stopped process $\mathbf{0}$, we replace it with the inactive process $\underline{\mathbf{0}}$ introduced in [9], which lets time pass according to the rule $\underline{\mathbf{0}} \overset{t}{\leadsto} \underline{\mathbf{0}}$. To avoid timelocks caused by restriction, for both calculi we opt for the following two-level syntax featuring only restriction at the top level (let $\mathbb{P}'_{\text{TCCS}}$ and $\mathbb{P}'_{\text{CIPA}}$ be the two resulting sets of closed and guarded process terms):

$$\begin{array}{rcl}
P' & ::= & P \mid P'\backslash L \\
P & ::= & \underline{\mathbf{0}} \mid \alpha.P \mid (t).P \mid P + P \mid P|P \mid P[\varphi] \mid X \mid \text{rec}\,X : P \\
Q' & ::= & Q \mid Q'\backslash L \\
Q & ::= & \text{nil} \mid a.Q \mid \text{wait}\,t.Q \mid Q + Q \mid Q|Q \mid Q[\varphi] \mid X \mid \text{rec}\,X : Q
\end{array}$$

Note that this modification does not limit the expressive power of the calculi. Suppose that a process P is made out of three subprocesses P_1, P_2, P_3 composed in parallel, such that P_1 has action a and the other two have action $\bar{a}$, but only P_1 and P_2 have to synchronize on a and $\bar{a}$. Normally, one would write $(P_1|P_2)\backslash\{a\}|P_3$, but this is forbidden by the revised syntax. However, the same effect can be obtained through $(P_1[\varphi]|P_2[\varphi]|P_3)\backslash\{b\}$, where relabeling function φ maps a to a fresh action b not occurring in any of the three subprocesses.

3.2 From Delays to Durations

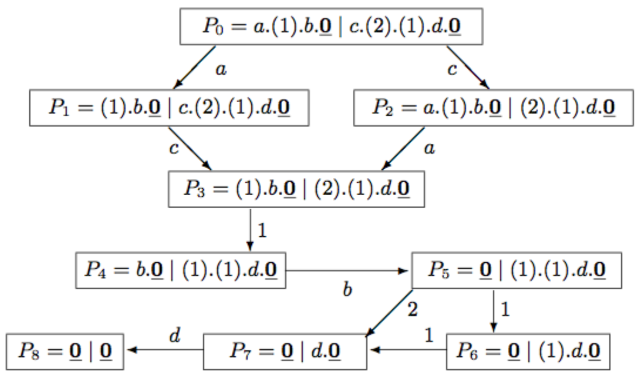
One of the major design decisions about the translation of (modified) TCCS into (modified) CIPA is how to assign durations to actions. In principle, it is desirable to be able to associate a suitable nonzero duration with every visible action occurring in a TCCS process that is not untimed. Unfortunately, in most cases this is not possible, as we now show.

Consider the TCCS process $a.(t_1).b.(t_2).\underline{\mathbf{0}}$. In this case, it is natural to interpret delay t_1 as the duration of a and delay t_2 as the duration of b, thus considering the occurrence of an instantaneous visible action of TCCS as the *beginning* of the corresponding durational action of CIPA (*initial view*). In the TCCS process $(t_1).a.(t_2).b.\underline{\mathbf{0}}$, the durations are as before, provided that the occurrence of an instantaneous visible action is considered as the *end* of the corresponding durational action (*final view*). Notice that, if $a = b$ but $t_1 \neq t_2$, the translation into a reasonably corresponding CIPA process would not be possible, unless, as noted in [5], we further extend the duration function for CIPA by admitting that different occurrences of the same action may have different durations.

Let us now examine the case in which there is not a precise pairing between actions and delays, like, e.g., in the TCCS process $(t_1).a.(t_2).\underline{\mathbf{0}}$. In this scenario, the duration of a can be either t_1 or t_2, but in any case a waiting is necessary to account for the delay that is not associated with a. The situation is even more complicated if we consider the TCCS process $a.(t_1).(t_2).b.\underline{\mathbf{0}}$. One option is to interpret $t_1 + t_2$ as the duration of a (initial view), with b having duration 0. The dual option is to interpret $t_1 + t_2$ as the duration of b (final view), with a having duration 0. In any case, the definition of the encoding would become technically involved, especially in the presence of recursion, due to the necessity of performing some lookahead. Moreover, there seems not to be any strong reason for choosing one option rather than the other.

Yet another option is to interpret t_1 as the duration of a (initial view) and t_2 as the duration of b (final view). This mixed option should be discarded because it disrupts equivalence preservation of the encoding. Indeed, the considered process is equivalent to the TCCS process $a.(t_2).(t_1).b.\underline{\mathbf{0}}$, while, under the assumption $t_1 \neq t_2$, the two corresponding CIPA processes are not equivalent to each other, because the a-transition of duration t_1 in the first CIPA process cannot be matched by the a-transition of duration t_2 of the second CIPA process.

Summing up, on the one hand there are TCCS delays that cannot be associated with any visible action, and hence have to be translated into CIPA



$P_0 = a.(1).b.\underline{0} \mid c.(2).(1).d.\underline{0}$
a
c
$P_1 = (1).b.\underline{0} \mid c.(2).(1).d.\underline{0}$
$P_2 = a.(1).b.\underline{0} \mid (2).(1).d.\underline{0}$
c
a
$P_3 = (1).b.\underline{0} \mid (2).(1).d.\underline{0}$
1
$P_4 = b.\underline{0} \mid (1).(1).d.\underline{0}$
b
$P_5 = \underline{0} \mid (1).(1).d.\underline{0}$
2
1
$P_8 = \underline{0} \mid \underline{0}$
d
$P_7 = \underline{0} \mid d.\underline{0}$
1
$P_6 = \underline{0} \mid (1).d.\underline{0}$

Fig. 1. Labeled transition system for P_0

waitings. On the other hand, it is not always possible to assign a nonzero duration to every TCCS visible action. In particular, this is impossible in the case of untimed TCCS processes. In this respect, it is also worth reminding that the two untimed TCCS processes $a.\underline{\mathbf{0}}|b.\underline{\mathbf{0}}$ and $a.b.\underline{\mathbf{0}} + b.a.\underline{\mathbf{0}}$ are always equivalent under the interleaving view of concurrency, while the two ideally corresponding CIPA processes $a.\text{nil}|b.\text{nil}$ and $a.b.\text{nil} + b.a.\text{nil}$ are equivalent to each other only if both a and b have duration zero.

As a consequence, for the sake of simplicity, uniformity, and semantics preservation, when encoding TCCS into CIPA we proceed as follows:

- Every TCCS action a will be translated into a CIPA action a with $\Delta(a) = 0$.
- The TCCS action τ will be translated into a CIPA waiting of duration 0 by allowing for waitings of the form wait $0.Q$ in the modified syntax of CIPA.
- Every TCCS delay t will be translated into a CIPA waiting of duration t.

3.3 Which Behavioral Equivalence Can Be Preserved?

While the encoding from CIPA to TCCS defined in [5] preserves strong timed bisimilarity, this cannot be the case for the reverse encoding from (modified) TCCS to (modified) CIPA.

Consider the two TCCS processes $a.(t_1).(t_2).b.\underline{\mathbf{0}}$ and $a.(t_1 + t_2).b.\underline{\mathbf{0}}$, which are equivalent to each other according to the strong timed bisimilarity of [8]. Their corresponding CIPA processes will be respectively $a.\text{wait}\, t_1.\text{wait}\, t_2.b.\text{nil}$ and $a.\text{wait}\,(t_1 + t_2).b.\text{nil}$, which are not equivalent to each other according to the strong timed bisimilarity defined in [5].

It is however worth pointing out that the two former processes are equivalent according to $\approx_{\text{TCCS}}$ and, most importantly, the two latter processes are equivalent according to $\approx_{\text{CIPA}}$. As a consequence, in this paper we have to restrict ourselves to weak timed bisimilarities when investigating semantics preservation for the reverse encoding.

4 The Reverse Encoding

In this section, we translate (modified) TCCS process terms into (modified) CIPA process terms and we show that the resulting encoding is fully abstract, in the sense that it preserves weak timed bisimilarity. The encoding is defined by induction on the syntactical structure of process terms.

Definition 5. *The encoding* $[\![\_]\!] : \mathbb{P}'_{\text{TCCS}} \to \mathbb{P}'_{\text{CIPA}}$ *is defined as follows:*

$$\begin{array}{rclc} [\![\underline{\mathbf{0}}]\!] & = & \text{nil} & [\![a.P]\!] = a.[\![P]\!] \\ [\![\tau.P]\!] & = & \text{wait}\,0.[\![P]\!] & [\![(t).P]\!] = \text{wait}\,t.[\![P]\!] \\ [\![P_1 + P_2]\!] & = & [\![P_1]\!] + [\![P_2]\!] & [\![P_1|P_2]\!] = [\![P_1]\!]|[\![P_2]\!] \\ [\![P\backslash L]\!] & = & [\![P]\!]\backslash L & [\![P[\varphi]]\!] = [\![P]\!][\varphi] \\ [\![X]\!] & = & X & [\![rec\,X : P]\!] = rec\,X : [\![P]\!] \end{array}$$

with $\Delta(a) = 0$ *for all* $a \in Act \setminus \{\tau\}$. ■

The states of the labeled transition systems underlying a TCCS process P and the corresponding $[\![P]\!]$ are strictly related. This relation is the key point that permits to show the main result of the paper stated in the forthcoming Thm. 1. Formally, to establish the relation, we need to add local clocks to $[\![P]\!]$. We let $\mathcal{E}[\![P]\!] \in \mathbb{KP}$ denote the encoded P process where a clock $0 \Rightarrow$ has been added to each sequential component.

Consider as an example the process $P_0 = a.(1).b.\underline{\mathbf{0}} \mid c.(2).(1).d.\underline{\mathbf{0}}$, whose transition system is depicted in Fig. 1. The transition system of $\mathcal{E}[\![P_0]\!]$ is shown in Fig. 2. It easy to see that, as far as only visible actions and zero-valued local clocks are concerned, the correspondence is one-to-one: $\mathcal{E}[\![P_i]\!] = K_i$, for $i = 0, 1, 2, 3$. However, in CIPA the wait t waitings, which produce τ actions, can be executed independently by one of the sequential components, which moves its own clock forward in time. In the meanwhile, the other components still have to execute actions "before" that time. This happens, for instance, in $K_1 \xrightarrow[1]{\tau@0} K'_1$; note, anyway, that K'_1 can still perform the visible action c at time 0.

Processes P_5, P_6 and P_7 are related by TCCS delay transitions. In the corresponding CIPA states, we can relate P_5 with K_5. The nil CIPA process lets any time pass for other sequential components. Thus, K_5 can be considered equivalent (at least with respect to $\approx_{\text{CIPA}}$) to $K'_5 = 2 \Rightarrow \text{nil} \mid 2 \Rightarrow \text{wait}\,1.d.\text{nil}$. In this way, $\mathcal{E}[\![P_6]\!] = 0 \Rightarrow \text{nil} \mid 0 \Rightarrow \text{wait}\,1.d.\text{nil}$ can be related to K'_5 by adjusting the clock values that are different in absolute value, but agree on the relative differences: $2 - 2 = 0 - 0 = 0$. Similarly, process P_7 can be related to process $K'_6 = 3 \Rightarrow \text{nil} \mid 3 \Rightarrow d.\text{nil}$ obtained from K_6.

Consider, finally, process P_4, derived from P_0. Its counterpart K_4 cannot perform any τ, but $\mathcal{E}[\![P_4]\!]$ has clock values not corresponding to K_4. Nevertheless, the b action is performed at the same time in both cases, i.e., with timestamp 1.

To formally treat the discrepancies mentioned above, it is convenient to define a structural congruence $\equiv$ over $\mathbb{KP}$ that permits to equate timed processes that respect, in their sequential components, the following two equations:

$$\begin{array}{rcl} m \Rightarrow \text{wait}\,n.P & = & m + n \Rightarrow P \\ n \Rightarrow \text{nil} & = & m \Rightarrow \text{nil} \end{array}$$

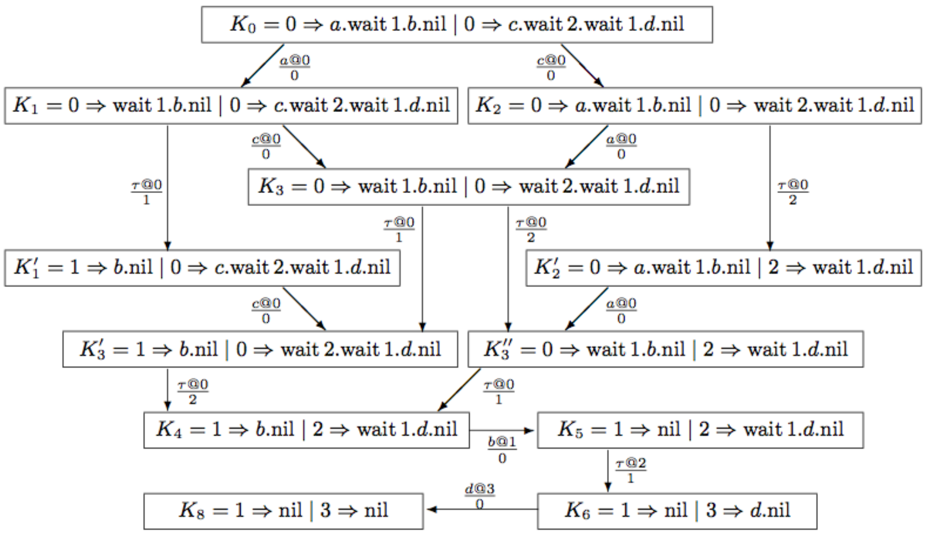

$K_0 = 0 \Rightarrow a.\text{wait}\,1.b.\text{nil} \mid 0 \Rightarrow c.\text{wait}\,2.\text{wait}\,1.d.\text{nil}$
$\frac{a@0}{0}$
$\frac{c@0}{0}$
$K_1 = 0 \Rightarrow \text{wait}\,1.b.\text{nil} \mid 0 \Rightarrow c.\text{wait}\,2.\text{wait}\,1.d.\text{nil}$
$K_2 = 0 \Rightarrow a.\text{wait}\,1.b.\text{nil} \mid 0 \Rightarrow \text{wait}\,2.\text{wait}\,1.d.\text{nil}$
$\frac{c@0}{0}$
$\frac{a@0}{0}$
$\frac{\tau@0}{1}$
$K_3 = 0 \Rightarrow \text{wait}\,1.b.\text{nil} \mid 0 \Rightarrow \text{wait}\,2.\text{wait}\,1.d.\text{nil}$
$\frac{\tau@0}{2}$
$\frac{\tau@0}{1}$
$\frac{\tau@0}{2}$
$K'_1 = 1 \Rightarrow b.\text{nil} \mid 0 \Rightarrow c.\text{wait}\,2.\text{wait}\,1.d.\text{nil}$
$K'_2 = 0 \Rightarrow a.\text{wait}\,1.b.\text{nil} \mid 2 \Rightarrow \text{wait}\,1.d.\text{nil}$
$\frac{c@0}{0}$
$\frac{a@0}{0}$
$K'_3 = 1 \Rightarrow b.\text{nil} \mid 0 \Rightarrow \text{wait}\,2.\text{wait}\,1.d.\text{nil}$
$K''_3 = 0 \Rightarrow \text{wait}\,1.b.\text{nil} \mid 2 \Rightarrow \text{wait}\,1.d.\text{nil}$
$\frac{\tau@0}{2}$
$\frac{\tau@0}{1}$
$K_4 = 1 \Rightarrow b.\text{nil} \mid 2 \Rightarrow \text{wait}\,1.d.\text{nil}$
$K_5 = 1 \Rightarrow \text{nil} \mid 2 \Rightarrow \text{wait}\,1.d.\text{nil}$
$\frac{b@1}{0}$
$\frac{\tau@2}{1}$
$\frac{d@3}{0}$
$K_8 = 1 \Rightarrow \text{nil} \mid 3 \Rightarrow \text{nil}$
$K_6 = 1 \Rightarrow \text{nil} \mid 3 \Rightarrow d.\text{nil}$

Fig. 2. Labeled transition System for $\mathcal{E}[\![P_0]\!]$

where $n, m \in \mathbb{N}$. It is easy to see that $\equiv$ implies $\approx_{\text{CIPA}}$. Moreover, following [5], we define $\mathit{wf} \subseteq \mathbb{KP} \times \mathbb{N}$ and $\mathit{up}\colon \mathbb{KP} \times \mathbb{N} \to \mathbb{KP}$. We let $\mathit{wf}(K, n)$ hold iff the local clocks of K can be decreased by n without any of them becoming negative, neglecting the components that are nil. If $\mathit{wf}(K, n)$, then $\mathit{up}(K, n)$ is the timed CIPA state K in which every local clock (apart from the nil components) has been decreased by n.

Using this notation, we get $\mathcal{E}[\![P_1]\!] = K_1 = \mathit{up}(K_1, 0) \equiv K'_1$, i.e., P_1 in Fig. 1 can be related to two timed states that are structural equivalent and, thus, $\approx_{\text{CIPA}}$ equivalent. Moreover, $\mathcal{E}[\![P_5]\!] = \mathit{up}(K_5, 1)$, $\mathcal{E}[\![P_6]\!] = \mathit{up}(K'_5, 1+1 = 2)$ where $K'_5 \equiv K_5$, $\mathcal{E}[\![P_7]\!] = \mathit{up}(K'_6, 2+1 = 3)$ where $K'_6 \equiv K_6$. Note that the subsequent times n in $\mathit{up}(\cdot, n)$ reflect the TCCS delay transitions. Finally, $\mathcal{E}[\![P_4]\!] = 0 \Rightarrow b.\text{nil} \mid 0 \Rightarrow \text{wait}\,1.\text{wait}\,1.d.\text{nil} \equiv 0 \Rightarrow b.\text{nil} \mid 1 \Rightarrow \text{wait}\,1.d.\text{nil} = \mathit{up}(K_4, 1)$.

Theorem 1. *Let $P_1, P_2 \in \mathbb{P}'_{\text{TCCS}}$. Then $P_1 \approx_{\text{TCCS}} P_2$ iff $[\![P_1]\!] \approx_{\text{CIPA}} [\![P_2]\!]$.* ■

5 Conclusions

In this paper, we have addressed the issues raised at the end of [5] and shown that it is possible, after applying certain modifications to the languages, to define a reverse semantics-preserving mapping from TCCS to CIPA. Unlike the direct encoding of [5] from CIPA to TCCS, which preserves strong timed bisimilarity, our reverse encoding can only preserve weak timed bisimilarity.

As future work, we want to investigate how to exploit our encoding of TCCS into CIPA together with the notion of compact representation of CIPA timed states introduced in [5], to achieve a better performance with respect to weak timed bisimilarity checking algorithms for TCCS. Moreover, we would like to extend the reverse mapping so to encode the idling operator and the weak choice operator of TCCS as well. Similar to [5], we also plan to investigate variants of our reverse encoding in which laziness or maximal progress is assumed in place of action urgency. Finally, continuing the work of [3], we would like to provide a uniform framework for comparing the various timed process calculi and timed models that have been proposed in the literature.

References

1. L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33:317–350, 1996.
2. J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3:142–188, 1991.
3. M. Bernardo and L. Tesei. Encoding timed models as uniform labeled transition systems. In *Proc. of the 10th European Performance Engineering Workshop (EPEW 2013)*, volume 8168 of *LNCS*, pages 104–118. Springer, 2013.
4. T. Bolognesi and F. Lucidi. LOTOS-like process algebras with urgent or timed interactions. In *Proc. of the 4th Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE 1991)*, volume C-2 of *IFIP Transactions*, pages 249–264, 1991.
5. F. Corradini. Absolute versus relative time in process algebras. *Information and Computation*, 156:122–172, 2000.
6. F. Corradini, W. Vogler, and L. Jenner. Comparing the worst-case efficiency of asynchronous systems with PAFAS. *Acta Informatica*, 38:735–792, 2002.
7. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117:221–239, 1995.
8. F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proc. of the 1st Int. Conf. on Concurrency Theory (CONCUR 1990)*, volume 458 of *LNCS*, pages 401–415. Springer, 1990.
9. F. Moller and C. Tofts. Behavioural abstraction in TCCS. In *Proc. of the 19th Int. Coll. on Automata, Languages and Programming (ICALP 1992)*, volume 623 of *LNCS*, pages 559–570. Springer, 1992.
10. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Proc. of the REX Workshop on Real Time: Theory in Practice*, volume 600 of *LNCS*, pages 526–548. Springer, 1991.
11. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.
12. J. Quemada, D. de Frutos, and A. Azcorra. TIC: A timed calculus. *Formal Aspects of Computing*, 5:224–252, 1993.
13. G. Reed and A. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.
14. I. Ulidowski and S. Yuen. Extending process languages with time. In *Proc. of the 6th Int. Conf. on Algebraic Methodology and Software Technology (AMAST 1997)*, volume 1349 of *LNCS*, pages 524–538. Springer, 1997.
15. W. Yi. CCS + time = an interleaving model for real time systems. In *Proc. of the 18th Int. Coll. on Automata, Languages and Programming (ICALP 1991)*, volume 510 of *LNCS*, pages 217–228. Springer, 1991.

Size-constrained 2-clustering in the plane with Manhattan distance

Alberto Bertoni, Massimiliano Goldwurm, Jianyi Lin, Linda Pini

Dipartimento di Informatica, Università degli Studi di Milano
Via Comelico 39/41, 20135 Milano – Italy,

Abstract. We present an algorithm for the 2-clustering problem with cluster size constraints in the plane assuming ℓ_1-norm, that works in $O(n^3 \log n)$ time and $O(n)$ space. Such a procedure also solves a full version of the problem, computing the optimal solutions for all possible constraints on cluster sizes. The algorithm is based on a separation result concerning the clusters of any optimal solution of the problem and on an extended version of red-black trees to maintain a bipartition of a set of points in the plane.

Keywords: algorithms and data structures, clustering, cluster size constraints, Manhattan distance.

1 Introduction

Clustering is one of the most used techniques in statistical data analysis and machine learning [5], with a wide range of applications in many areas like pattern recognition, bioinformatics and image processing. Clustering consists in partitioning data into groups, called *clusters*, that are required to be homogeneous and well-separated according to a data similarity measure [9]. A natural representation of the data elements is by means of points in a d-dimensional space equipped with a suitable distance function that determines the similarity measure. We study this very usual case, called *distance clustering.*

A central problem in distance clustering is the well-known Minimum Sum-of-Squares Clustering (MSSC), also called Variance-based Clustering [2, 10]. The MSSC problem requires to find a k-partition $\{A_1, ..., A_k\}$ of a given set $X = \{x_1, ..., x_n\} \subset \mathbb{R}^d$ of n points that minimizes the weight

$$W(A_1, ..., A_k) = \sum_{i=1}^{k} \sum_{x \in A_i} ||x - C_{A_i}||_2^2$$

where $||\cdot||_2$ denotes the Euclidean norm and C_{A_i} is the mean point of the cluster A_i, defined by

$$C_{A_i} = \operatorname*{argmin}_{\mu \in \mathbb{R}^d} \sum_{x \in A_i} ||\mu - x||_2^2 = \frac{1}{|A_i|} \sum_{x \in A_i} x.$$

MSSC is difficult in general; indeed for an arbitrary dimension d (given in the instance) the problem is NP-hard even if the number of clusters is fixed to $k = 2$ [1, 8]; the same occurs if k is arbitrary and the dimension is fixed to $d = 2$ [14]. However there's a well-known heuristic for finding an approximate solution of MSSC, called k-Means [13], which is known to be usually very fast, but can require exponential time in the worst case [17].

Moreover, a known variant of MSSC, called k-Medians Problem, is given by substituting the squared-euclidean norm $||\cdot||_2^2$ with the ℓ_1-norm $||\cdot||_1$ [15].

In many applications, often people have some information on the clusters [3]: including this information into traditional clustering algorithms can increase the clustering performance. Problems that include such background information are called *constrained clustering* problems and are split into two classes. On the one hand, clustering problems with point-level constraints typically comprise a set of must-link constraints or cannot-link constraints [18], defining pairs of point that, respectively, must be or cannot be grouped in the same cluster. On the other hand, clustering problems with cluster-level constraints [6, 16] provides constraints concerning the size of the resulting clusters. For example, in [19] cluster size constraints are used for improving clustering accuracy; this approach, for instance, allows one to avoid extremely small or large clusters yielded by classical clustering techniques.

In this work we study the 2-clustering problem in the plane, with cluster size constraints, assuming ℓ_1-norm. The relevance of the 2-clustering problems is also due to the wide spread of hierarchical clustering techniques [11, Sec.14.3.12], that repeatedly apply the 2-clustering as the key step. This leads to natural size-constrained versions of so-called divisive hierarchical clustering.

We recall that the 2-clustering problem with cluster size constraints has been studied in [12, 4], where it is shown that in dimension 1 the problem is solvable in polynomial time for every norm ℓ_p with integer $p \geq 1$, while there is some evidence that the same result does not hold for non-integer p. Moreover, it is also known that for arbitrary dimension d the same problem is NP-hard even assuming equal sizes of the two clusters.

Here we prove that, assuming dimension 2 and ℓ_1-norm, the 2-clustering problem with cluster size constraints can be solved in $O(n^3 \log n)$ time and $O(n)$ space. Our procedure actually solves a full version of the problem, computing the optimal solutions for all possible sizes of cluster. Clearly this yields the solution to the general unconstrained 2-clustering problem in the plane with ℓ_1-norm.

The algorithm is based on a separation result stating that, in our hypotheses, the clusters of any optimal solution are separated by curves of some special forms, which can be analysed in a suitable linear order. The algorithm also make use of a particular extension of red-black trees to maintain a bipartition of a set of points in the plane.

We remark that in this work we propose an efficient method for obtaining a solution that is globally optimal, instead of a locally optimal solution as yielded by various heuristics [3, 6, 19].

2 Problem definition

In this section we introduce some basic notions of distance-based clustering problems on the plane assuming the Manhattan norm (also called ℓ_1-norm), which is defined by $\|x\|_1 = |x_1| + |x_2|$ for any $x = (x_1, x_2) \in \mathbb{R}^2$.

Given a set $X \subset \mathbb{R}^2$ of n points in the plane, a *k-partition* of X is a family $\{A_1, A_2, ..., A_k\}$ of k nonempty subsets of X such that $\cup_{i=1}^k A_i = X$ and $A_i \cap A_j = \emptyset$, for $i \neq j$. Each A_i is called *cluster*. The *centroid* C_A of a cluster $A \subset X$ is

$$C_A = \underset{\mu \in \mathbb{R}^2}{\operatorname{argmin}} \sum_{x \in A} \|x - \mu\|_1 = \underset{\mu \in \mathbb{R}^2}{\operatorname{argmin}} \sum_{x \in A} (|x_1 - \mu_1| + |x_2 - \mu_2|)$$

Since the objective function in the minimization is not strictly convex, the centroid C_A is not necessarily unique. Indeed it is well-known that the centroid C_A is the component-wise median, i.e. the abscissa (ordinate) of the centroid is the median of the abscissae (ordinates) of the points in A. The *weight* $W(A)$ of a cluster A is

$$W(A) = \sum_{x \in A} \|x - C_A\|_1$$

while the *weight* of a k-partition $\{A_1, A_2, ..., A_k\}$ (also called k-*clustering*) is

$$W(A_1, A_2, \cdots, A_k) = \sum_1^k W(A_i).$$

The classical *Clustering Problem* in the plane assuming the Manhattan norm is stated as follows.

Definition 1 (Clustering Problem in $\mathbb{R}^2$ under ℓ_1-norm). *Given a point set $X \subset \mathbb{R}^2$ of cardinality n and an integer k, $1 < k < n$, find a k-clustering $\{A_1, A_2, ..., A_k\}$ that minimizes the weight $W(A_1, A_2, \cdots, A_k) = \sum_1^k W(A_i)$.*

Note that here k is included in the instance of the problem. If k is fixed the problem is called k-Clustering in $\mathbb{R}^2$. Our main contribution in this paper concerns the 2-Clustering in $\mathbb{R}^2$ under ℓ_1-norm.

We are interested in a version of clustering problem where the cluster sizes are constrained. Formally, the problem can be stated as follows:

Definition 2 (Size Constrained Clustering Problem in $\mathbb{R}^2$ under ℓ_1-norm). *Given a point set $X \subset \mathbb{R}^2$ of cardinality n, an integer $k > 1$ and k positive integers $m_1, m_2, ..., m_k$ such that $\sum_1^k m_i = n$, find a k-clustering $\{A_1, A_2, ..., A_k\}$ with $|A_i| = m_i$ for $i = 1, ..., k$, that minimizes the weight $W(A_1, A_2, \cdots, A_k) = \sum_1^k W(A_i)$.*

We denote this problem by SCC-2 (under ℓ_1-norm). We stress that in the SCC-2 problem the integers n, k, $m_1, \ldots, m_k$ are part of the instance. On the contrary, if k is fixed and does not belong to the instance, the problem is denoted by k-SCC-2.

In the following sections we present an algorithm for the 2-Clustering problem in $\mathbb{R}^2$ under ℓ_1-norm, which solves at the same time the *full* version of the 2-SCC-2 problem, i.e. that yields the solutions to all 2-SCC-2 problems for every value of $m_1 \in \{1, 2, \dots, \lfloor n/2 \rfloor\}$ and the same point set $X = \{x_1, x_2, ..., x_n\} \subset \mathbb{R}^2$ in input. The procedures works in $O(n^3 \log n)$ time and is based on a separation result concerning the optimal solutions of 2-Clustering in the plane, presented in Section 3, and on an augmented version of red-black trees used to maintain bipartitions of a set of real points, described in Section 4.

3 Separation results

In this section we present a separation result concerning the optimal solution of 2-SCC-2 problem under ℓ_1-norm. It turns out that the clusters of the optimal solutions of this problem are separated by plane curves of 8 different types, we will call C_i and S_i, $i = 1, 2, 3, 4$, respectively. Similar results are obtained in [4] for higher dimensional spaces and ℓ_p-norms, with $p > 1$.

First, we show that in an optimal 2-clustering the swapping of two elements in different clusters does not decrease the solution value.

Lemma 1 (Swapping Lemma). *Let $\{A, B\}$ be an optimal solution of the 2-SCC-2 problem. Then, for any $a \in A$ and $b \in B$, it holds*

$$||a - C_A||_1 + ||b - C_B||_1 \leq ||a - C_B||_1 + ||b - C_A||_1 .$$

Proof. For the sake of simplicity, we omit the subscript 1 in denoting norm. By contradiction, let us assume that for some $a \in A$ and $b \in B$

$$||a - C_A|| + ||b - C_B|| > ||a - C_B|| + ||b - C_A|| .$$

Then, the weight of the optimal solution is

$$\begin{aligned} W(A,B) &= \sum_{x \in A} ||x - C_A|| + \sum_{x \in B} ||x - C_B|| = \\ &> \sum_{x \in A \smallsetminus \{a\}} ||x - C_A|| + ||b - C_A|| + \sum_{x \in B \smallsetminus \{b\}} ||x - C_B|| + ||a - C_B|| = \\ &= \sum_{x \in A \smallsetminus \{a\} \cup \{b\}} ||x - C_A|| + \sum_{x \in B \smallsetminus \{b\} \cup \{a\}} ||x - C_B|| . \end{aligned}$$

Now, it is clear that the 2-clustering $\{A', B'\}$, with $A' = A \smallsetminus \{a\} \cup \{b\}$ and $B' = B \smallsetminus \{b\} \cup \{a\}$, is a feasible solution since $|A'| = |A|$ and $|B'| = |B|$. Furthermore, by definition of centroid it holds:

$$\sum_{x \in A'} ||x - C_A|| + \sum_{x \in B'} ||x - C_B|| \geq \sum_{x \in A'} ||x - C_{A'}|| + \sum_{x \in B'} ||x - C_{B'}|| = W(A', B')$$

and hence $W(A, B) > W(A', B')$. However, this is in contradiction since $\{A, B\}$ is an optimal solution. □

We are now able to obtain the general form of the equation for the curves separating the clusters in an optimal 2-clustering.

Theorem 1 (Separation Result). *In an optimal solution $\{A, B\}$ of the 2-SCC-2 problem, the clusters A and B are separated by a curve of equation*

$$||z - \alpha||_1 - ||z - \beta||_1 = g \tag{1}$$

where the variable z ranges over $\mathbb{R}^2$, while $\alpha, \beta \in \mathbb{R}^2$ and $g \in \mathbb{R}$ are suitable constants.

Proof. Let C_A and C_B be the centroids of A and B respectively. Then, by Lemma 1 the inequality

$$||a - C_A||_1 - ||a - C_B||_1 \leq ||b - C_A||_1 - ||b - C_B||_1$$

is satisfied for all $a \in A$ and $b \in B$, and hence it also holds by taking the maximum over a's on the left and the minimum over b's on the right:

$$c_A := \max_{a \in A}\{||a - C_A||_1 - ||a - C_B||_1\} \leq \min_{b \in B}\{||b - C_A||_1 - ||b - C_B||_1\} =: c_B$$

Therefore, choosing $g \in [c_A, c_B]$, all the points of A and B are contained in the region defined respectively by $||z - C_A||_1 - ||z - C_B||_1 \leq g$ and $||z - C_A||_1 - ||z - C_B||_1 \geq g$. The common boundary of the two regions has equation $||z - C_A||_1 - ||z - C_B||_1 = g$, and thus the proof is concluded by setting $\alpha = C_A$ and $\beta = C_B$. $\square$

Remark 1. It is noteworthy to observe that both the Swapping Lemma and the Separation Result can be extended to the case of ℓ_p-norm with any $p \geq 1$ and dimension $d \geq 2$, provided the norm is raised to the p-th power [12].

Setting the symbols $z = (x, y)$, $\alpha = (c, d)$, $\beta = (e, f)$, equation (1) becomes

$$|x - c| - |x - e| + |y - d| - |y - f| = g$$

It is clear that it always represents a connected curve on the plane, which may have some sharp corner points, i.e. points where one or both partial derivatives do not exist. A rather standard study of this equation, conducted by considering all possible values of c, d, e, f, g, leads to derive 8 types of separating curves, we denote by symbols C_1, C_2, C_3, C_4 and S_1, S_2, S_3, S_4, according to their shape. The proof is here omitted for lack of space. Such 8 types of curves are depicted in Figures 1–4.

4 Bipartition red-black trees

In this section we describe a natural data structure to maintain a bipartition of a set of real numbers, with possible multiple values, able to test membership, to execute insertion and deletion, and to perform two further operations: selecting the i-th smallest element in either of the two clusters of the bipartition, for any integer i, and moving an element from one cluster to the other.

Let X be a finite multiset of real numbers, i.e. a finite set X of reals where any value may occur several times, represented as a finite ordered sequence

$$X = (x_1, x_2, \ldots, x_n), x_i \in \mathbb{R} \text{ for every } i, \text{ and } x_1 \leq x_2 \leq \cdots \leq x_n.$$

Moreover, consider a bipartition $\{A, B\}$ of X, i.e. two non-empty subsets $A \subseteq X$, $B \subseteq X$ such that $A \cap B = \emptyset$ and $A \cup B = X$. In the following A and B are called clusters. Operations Member, Insert and Delete can be defined in order to specify the cluster the elements belong to. More precisely, for every $z \in \mathbb{R}$, Member$(z) = (j_1, j_2)$, where j_1 (resp. j_2) is the number of x_i's in A (resp. in B) such that $x_i = z$, Insert(z, A) adds a new element z to A, Insert(z, B) does the same to B, while Delete(z, A) and Delete(z, B) execute the delete operations.

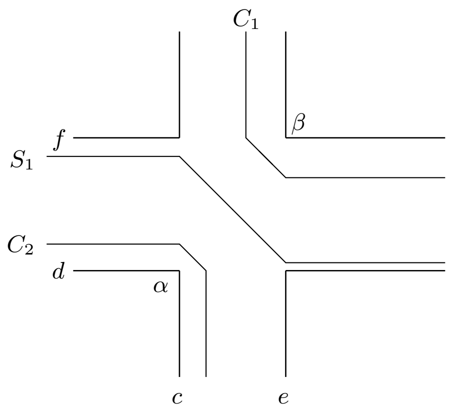


Figure 1

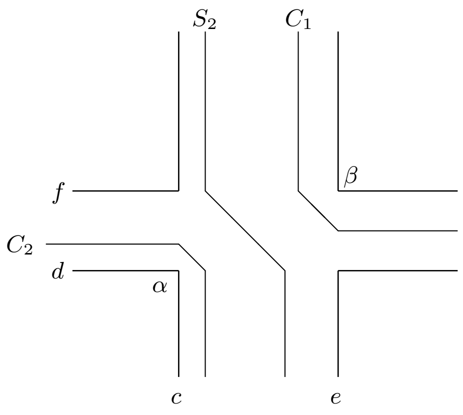


Figure 2

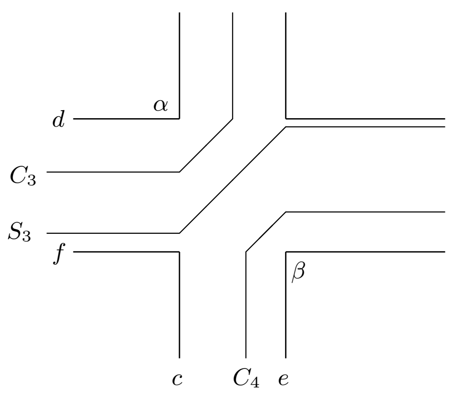


Figure 3

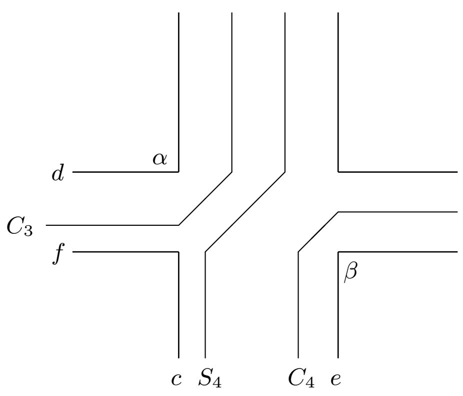


Figure 4

Moreover, for every $i \in \{1, 2, \ldots, n\}$, we define

$$\text{Select}_A(i) = \begin{cases} a & \text{if } i \leq |A| \text{ and } a \text{ is the } i\text{-th smallest element of } A \\ \perp & \text{if } i > |A| \end{cases}$$
$$\text{Select}_B(i) = \begin{cases} b & \text{if } i \leq |B| \text{ and } b \text{ is the } i\text{-th smallest element of } B \\ \perp & \text{if } i > |B| \end{cases} .$$

Finally, for any $z \in \mathbb{R}$, if A contains an element $x_i = z$ then $\text{Move}_{AB}(z)$ deletes a value z from A and adds it to B, otherwise $\text{Move}_{AB}(z) = \perp$. The operation $\text{Move}_{BA}(z)$ is defined symmetrically.

A natural data structure, able to executes the above operations in time logarithmic with respect to $n = |X|$, is an augmented version of the well-known red-black trees [7]. Formally, we define a *Bipartition Red-Black Tree* for a bipartition $\{A, B\}$ of X as a binary tree T such that: *i*) all internal nodes have both the left and the right son, *ii*) every node is red or black, *iii*) the root and the leaves are black, *iv*) if a node is red its sons are black, and *v*) for every node v all simple paths from v to any leaf have the same number of black nodes. Further, the distinct elements of X (called keys) are assigned to the internal nodes in a bijective correspondence, respecting the standard rule of binary search trees, and the leaves are empty (usually represented by "nil"). Moreover, T satisfies the following two conditions:

1. every internal node v contains a triple $(z, m_A(v), m_B(v))$, where z is the key assigned to v, and $m_A(v)$ (respectively, $m_B(v)$) is the number of elements x_i belonging to A (resp. B) such that $x_i = z$;
2. every internal node v contains the pair of integers $(c_A(v), c_B(v))$, where $c_A(v)$ (resp. $c_B(v)$) is the number of elements of X belonging to A (resp. B) whose value is assigned to a vertex in the subtree rooted at v.

For each internal node v, we denote by *key*(v), *left*(v) and *right*(v) the key contained in v, the left son and the right son of v, respectively. Clearly, v can be implemented as a record including the values $m_A(v)$, $m_B(v))$, $c_A(v)$, $c_B(v)$.

Procedures Member, Insert and Delete can be defined as in the standard red-blacks trees [7] with obvious changes. Clearly, Insert and Delete have to update the values $m_A(v)$, $m_B(v))$, $c_A(v)$, $c_B(v)$, for all nodes v on the path from the root to the vertex resulting from the binary search.

Operations Select and Move can be executed as follows. To compute $\text{Select}_A(i)$ one checks whether $i \leq c_A(r)$, where r is the root of T: in the affirmative case a standard searching procedure (we avoid to define for lack of space) is called, otherwise the value $\perp$ is returned. $\text{Select}_B(i)$ is defined in a similar way.

To compute $\text{Move}_{AB}(z)$ one first determines the node v containing z by calling a rather usual procedure, then the commands

$$m_A(v) = m_A(v) - 1 \quad \text{and} \quad m_B(v) = m_B(v) + 1$$

are executed. Procedure $\text{Move}_{BA}(z)$ is defined symmetrically.

It is clear that all previous procedures can be executed in $O(\log n)$ time.

5 Updating the weight of clusters

Our main result, presented in the next section, is based on a procedure that updates a current bipartition by moving one point from a cluster to the other. Here we want to show how to update the centroids and the weights of the clusters of a current bipartition after executing such a moving operation.

To this end we first deal with the problem of updating centroid and weight of a set of real numbers subject to insert and delete operations. The results will be easily extended to a set of points in $\mathbb{R}^2$.

Let A be a cluster in $\mathbb{R}$, i.e. a finite sequence of ordered real numbers, $A = \{a_1, a_2, \dots, a_k\}$, where $a_1 \leq a_2 \leq \cdots \leq a_k$. Assuming ℓ_1-norm it is well-known that the centroid of A is the median of the set, that is the value

$$M(A) = \begin{cases} a_{\frac{k+1}{2}} & \text{if } k \text{ is odd} \\ \frac{1}{2}\left(a_{\frac{k}{2}} + a_{\frac{k}{2}+1}\right) & \text{if } k \text{ is even} \end{cases}$$

Thus, if A is a cluster of a bipartition (of a finite multiset $X \subset \mathbb{R}$ of n elements), maintained by a Bipartition Red-Black Tree, then $M(A)$ can be computed in $O(\log n)$ time by calling procedure $\text{Select}_A(j)$ for suitable j's.

Now, let us recall the definition of the left and right part of the weight of A; given $m = \frac{k+1}{2}$, we have

$$L(A) := \sum_{1 \leq i < m} a_i \ , \qquad R(A) := \sum_{m < i \leq k} a_i$$

It is not difficult to see that the weight of A, given by $W(A) = \sum_{i=1}^{k} |a_i - M(A)|$, can be computed as the difference between $R(A)$ and $L(A)$.

Proposition 1. *For every ordered sequence of real numbers $A = \{a_1, a_2, \dots, a_k\}$, where $a_1 \leq a_2 \leq \cdots \leq a_k$, we have*

$$W(A) = R(A) - L(A)$$

Proof. If k is even then $m = \frac{k+1}{2}$ is not integer and hence

$$\begin{aligned} W(A) = \sum_{i=1}^{k} |a_i - M(A)| &= \sum_{1 \leq i < m} (M(A) - a_i) + \sum_{m < i \leq k} (a_i - M(A)) \\ &= -L(A) + R(A) \end{aligned}$$

If k is odd the reasoning is almost the same. □

As a consequence, in order to maintain $W(A)$ it is sufficient to update $L(A)$ and $R(A)$. This can be done by a straightforward procedure that implements, as a chain of if-then-else instructions, the rules for computing the left and right part of the weight of a current cluster upon insertion and deletion. For lack of space here we omit their cumbersome formal description. These rules together with

Proposition 1, define procedures UpdateW-ins(A, x) and UpdateW-del(A, a_j), which update the weight of a current cluster $A \subset \mathbb{R}$, respectively upon insertion and deletion of an element of value $x \in \mathbb{R}$.

They can be executed by using a Bipartition Red-Black tree with the corresponding Select operation, introduced in the previous section. Note that, for updating both $L(A)$ and $R(A)$, the Select procedure is necessary to find in the tree the i-th value of A when required. As a consequence also UpdateW-ins(A, x) and UpdateW-del(A, a_j) can be executed in logarithmic time.

6 Main algorithm

In this section we apply the previous results, in particular the separation property of Section 3, to define an algorithm for the 2-SCC-2 problem in the full version, i.e. for all possible sizes of cluster. Thus the same algorithm also yields a solution of the general 2-Clustering problem in the plane.

By our Theorem 1, the optimal solution of a 2-SCC-2 problem for a set $X \subset \mathbb{R}^2$ of n points and a cluster size m, consists of a bipartition (A, B), with $|A| = m$ and $|B| = n - m$, where A and B are separated by a curve of the form C_i or S_i, $i = 1, 2, 3, 4$.

Note that each curve of type C_i can be obtained from a curve of any other form C_j, $j \neq i$, by means of a simple rotation of the plane. Similarly each curve of type S_i can be obtained from another curve of any different type S_j by means of a rotation and/or a reflection through a line.

For these reasons we treat in detail only the case of bipartitions separated by a curve of type C_1. The cases of curves of type C_2, C_3 and C_4 are solved by the same algorithm through a rotation of the input points. The same algorithm can be easily adapted to the cases of separating curves of type S_i, $i = 1, 2, 3, 4$.

Let $X = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ be a set of n points and let us set $p_i = (x_i, y_i)$, for any $i = 1, 2 \ldots, n$. Without loss of generality we may assume that all p_i's lie in the first quadrant, i.e. $x_i > 0$ and $y_i > 0$ for all i.

We define a procedure that searches an optimal bipartition of X whose clusters are separated by a curve of type C_1. The procedure first sorts X according with 3 parameters: the two coordinates and their sum, i.e. the values x_i's, y_i's and $x_i + y_i$'s, respectively. Then the elements of X are processed by three for-loops, each nested into the other, corresponding to the three parameters considered above. The algorithm evaluates all possible bipartitions of X whose clusters are separated by curves of type C_1 and, for every admissible cluster size, only the bipartition with minimum weight is maintained. These bipartitions are considered in a linear order and, at each step, a new bipartition of X is obtained from the current one $\{A, B\}$ by swapping a point from cluster B to cluster A.

Such a current bipartition $\{A, B\}$ is maintained by two bipartition red-black trees: one for the 2-clustering $\{A_x, B_x\}$ of the set $\{x_i \mid p_i = (x_i, y_i) \in X\}$, where $A_x = \{x_i \mid p_i \in A\}$ and $B_x = \{x_i \mid p_i \in B\}$, and the other for the 2-clustering (A_y, B_y) of $\{y_i \mid p_i = (x_i, y_i) \in X\}$, where $A_y = \{y_i \mid p_i \in A\}$ and $B_y = \{y_i \mid p_i \in B\}$.

An Update subroutine can be defined which, for an input (A, B, p_i), where $\{A, B\}$ is the current bipartition of X and $p_i \in B$, computes the new bipartition moving p_i from B to A and returns the weights of its clusters. The subroutine applies the procedures described in Sections 4 and 5 and it is defined by the following scheme:

```
Procedure Update($A, B, p_i$)
begin
        Move$_{B_x A_x}(x_i)$
        Move$_{B_y A_y}(y_i)$
        $w(A_x) :=$ UpdateW-ins$(A_x, x_i)$
        $w(B_x) :=$ UpdateW-del$(B_x, x_i)$
        $w(A_y) :=$ UpdateW-ins$(A_y, y_i)$
        $w(B_y) :=$ UpdateW-del$(B_y, y_i)$
        return $(w(A_x) + w(A_y), w(B_x) + w(B_y))$
end
```

By the discussion presented in Sections 4 and 5 also this subroutine only requires $O(\log n)$ time.

Thus, the overall algorithm is described in detail by the procedure given below, where three bipartitions are actually maintained, denoted by (A, B), (A', B') and (A'', B''), respectively. Each of them need two Bipartition Red-Black trees, one for the abscissae and the other for the ordinates.

Moreover, instruction Sort$_x$ (resp., Sort$_y$ and Sort$_{x+y}$) yields an ordered list of the indices of the points in X, sorted in non-decreasing order with respect to the values x_i (resp., y_i and $x_i + y_i$).

Taking into account the previous discussion, it is easy to see that the algorithm works in $O(n^3 \log n)$ time; moreover, it has a space complexity $O(n)$ since each Bipartition Red-Black tree only requires linear space.

```
Procedure Full-biclustering($p_1, p_2, \dots, p_n$)
begin
        $(i_1, i_2, \dots, i_n) :=$ Sort$_x(1, 2, \dots, n)$
        $(j_1, j_2, \dots, j_n) :=$ Sort$_y(1, 2, \dots, n)$
        $(k_1, k_2, \dots, k_n) :=$ Sort$_{x+y}(1, 2, \dots, n)$
        $A := \emptyset$ ; $w(A) := 0$
        $B := \{p_1, p_2, \dots, p_n\}$
        compute the weight $w(B)$ of cluster $B$
        $i_0 := 0$ ; $j_0 := 0$ ; $x_0 := 0$ ; $y_0 := 0$
        for $r = 0, 1, \dots, n$ do
            $A' := A$ ; $B' := B$ ; $w(A') := w(A)$ ; $w(B') := w(B)$
            for $s = 0, 1, \dots, n$ do
                $A'' := A'$ ; $B'' := B'$ ; $w(A'') := w(A')$ ; $w(B'') := w(B')$
                for $k = k_1, k_2, \dots, k_n$ do
                    if $x_k \geq x_{i_r} \wedge y_k \geq y_{j_s}$ then
                        $(w(A''), w(B'')) =$ Update$(A'', B'', p_k)$
                $h := \min(|A''|, |B''|)$
                $z := w(A'') + w(B'')$
                if $W[h] > z$ then $\begin{cases} W[h] := z \\ \Pi[h] := (i_r, j_s, k) \end{cases}$
            if $s \neq n \wedge x_{j_{s+1}} \geq x_{i_r}$ then
                $(w(A'), w(B')) = \text{Update}(A', B', p_{j_{s+1}})$
                $h := \min(|A'|, |B'|)$
                $z := w(A') + w(B')$
                if $W[h] > z$ then $\begin{cases} W[h] := z \\ \Pi[h] := (i_r, j_{s+1}, 0) \end{cases}$
        if $r \neq n$ then
            $(w(A), w(B)) = \text{Update}(A, B, p_{i_{r+1}})$
            $h := \min(|A|, |B|)$
            $z := w(A) + w(B)$
            if $W[h] > z$ then $\begin{cases} W[h] := z \\ \Pi[h] := (i_{r+1}, 0, 0) \end{cases}$
    return $(W, \Pi)$
end
```

7 Conclusions

In this paper we have shown a $O(n^3 \log n)$ time algorithm for the size-constrained 2-Clustering problem in the plane with ℓ_1-norm. It is clear that our results strictly depend on those conditions and it is not evident (at least at the present time) whether they hold under different hypotheses or for more general problems. Moreover, being our algorithm exact and working in polynomial time, we do not think it is necessary to produce here, in our hypotheses (2-clustering in $\mathbb{R}^2$ with ℓ_1-norm), an experimental comparison with traditional heuristics like k-Means, which instead yields an approximate solution, tackles a more general problem with larger number of clusters, and works in exponential time in the worst case.

However, we think that the present contribution could be the starting point for a further analysis, still based on the separation result introduced in [4], of more general clustering problems, with dimension greater than 2, larger number of clusters, different norms and other constraints. It is clear that such a research direction should also include an experimental activity that plans a comparison with classical heuristics.

We also remark we have already obtained recently some results under other hypotheses. In particular, in a companion paper we present more efficient algorithms for solving exactly the same problem with Euclidean norm, based on rather different techniques of computational geometry. More precisely, it turns out that the 2-Clustering problem in $\mathbb{R}^2$ under Euclidean norm, with size constraint k, can be solved by an exact algorithm in $O(n\sqrt[3]{k}\log^2 n)$ time, while the full version of the same problem (i.e. for all $k = 1, 2, \ldots, \lfloor n/2 \rfloor$) is solvable in $O(n^2 \log n)$ time.

References

1. D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–249, 2009.
2. D. Aloise and P. Hansen. On the complexity of minimum sum-of-squares clustering. Technical report, Les Cahiers du GERAD, 2007.
3. S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications.* Chapman and Hall/CRC, 2008.
4. A. Bertoni, M. Goldwurm, J. Lin, and F. Saccà. Size Constrained Distance Clustering: Separation Properties and Some Complexity Results. *Fundamenta Informaticae*, 115(1):125–139, 2012.
5. C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.
6. P. S. Bradley, K. P. Bennett, and A. Demiriz. Constrained K-Means Clustering. Technical Report MSR-TR-2000-65, Miscrosoft Research Publication, May 2000.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001.
8. S. Dasgupta. The hardness of k-means clustering. Technical Report CS2007-0890, Dept. Computer Sc. and Eng., Univ. of California, San Diego, 2007.
9. W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53(284):789–798, 1958.
10. S. Hasegawa, H. Imai, M. Inaba, and N. Katoh. Efficient algorithms for variance-based k-clustering. In *Proceedings of Pacific Graphics '93*, pages 75–89, 1993.
11. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, 2nd edition, 2009.
12. J. Lin. *Exact algorithms for size constrained clustering.* PhD Thesis, Università degli Studi di Milano. Ledizioni Publishing, Milano, 2013.
13. J. B. MacQueen. Some method for the classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Structures*, pages 281–297, 1967.
14. M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012.
15. K. Sabo. Center-based l1-clustering method. *International Journal of Applied Mathematics and Computer Science*, 24(1):7–225, 2014.
16. A. Tung, J. Han, L. Lakshmanan, and R. Ng. Constraint-Based Clustering in Large Databases. In J. Van den Bussche and V. Vianu, editors, *Database Theory ICDT 2001*, volume 1973 of *LNCS*, pages 405–419. Springer Berlin/Heidelberg, 2001.
17. A. Vattani. K-means requires exponentially many iterations even in the plane. In *Proceedings of the 25th Symposium on Computational Geometry (SoCG)*, 2009.
18. K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proc. of the 17th Intl. Conf. on Machine Learning*, pages 1103–1110, 2000.
19. S. Zhu, D. Wang, and T. Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883–889, 2010.

Graphs of edge-intersecting and non-splitting paths [⋆]

Arman Boyacı[1], Tınaz Ekim[1], Mordechai Shalom[2], and Shmuel Zaks[3]

[1] Department of Industrial Engineering, Bogazici University, Istanbul, Turkey
[arman.boyaci,tinaz.ekim]@boun.edu.tr
[2] TelHai College, Upper Galilee, 12210, Israel cmshalom@telhai.ac.il [⋆⋆]
[3] Department of Computer Science, Technion, Haifa, Israel.
zaks@cs.technion.ac.il

Abstract. The families of Edge Intersection Graphs of Paths in a tree (resp. in a grid) EPT (resp. EPG) are well studied graph classes. Recently we introduced the class of graphs of Edge-Intersecting and Non-Splitting Paths in a Tree (ENPT) [2]. In this model, two vertices are adjacent if they represent two intersecting paths of a tree whose union is also a path. In this study we generalize this graph class by allowing the host graph to be an arbitrary graph. These are the graphs of Edge-Intersecting and Non-Splitting Paths ENP. Although the Edge Intersection Graphs of Paths in an arbitrary graph includes all graphs, we show that this is not the case for ENP. We also show that the class ENP coincides with the family of graphs of Edge-Intersecting and Non-Splitting Paths in a Grid (ENPG). Following similar studies for EPG graph class, we study the implications of restricting the number of bends in the grid, of the individual paths. We show that restricting the bend number also restricts the graph class. Specifically, by restricting the number of bends one gets an infinite sequence of classes such that every class is properly included in the next one.

1 Introduction

1.1 Background

The intersection graph of a family of sets is a graph in which every vertex corresponds to a set in the family, and two vertices of the graph are adjacent if and only if their corresponding set have a non-empty intersection. It is easy to see that every graph is an intersection graph. Therefore, it makes sense to study intersection graphs only when one focuses on specific families of sets. The family of paths on graphs is a commonly studied family of sets. To distinguish the graph on which the paths are defined, from the resulting intersection graph, this graph

[⋆] This work was supported in part by TUBITAK PIA BOSPHORUS Grant No. 111M303.
[⋆⋆] Currently in the Department of Industrial Engineering, Bogazici University, Istanbul, Turkey, supported by TUBITAK 2221 Programme.

is termed the *host* graph. Given a host graph H and a set $\mathcal{P}$ of paths in H, the Edge Intersection Graph of Paths (EP graph) of $\mathcal{P}$ is denoted by $\textsc{Ep}(\mathcal{P})$. $\textsc{Ep}(\mathcal{P})$ contains a vertex for each path in $\mathcal{P}$, and it contains an edge between two vertices if the corresponding two paths intersect in at least one edge. A graph G is EP if there exist a graph H and a set $\mathcal{P}$ of paths in H such that $G = \textsc{Ep}(\mathcal{P})$. In this case we say that $\langle H, \mathcal{P} \rangle$ is an EP representation of G. We also denote by EP the family of all graphs G that are EP.

EP graphs have applications in many areas, and in particular in communication networks. Consider a communication network, and routes of messages to be delivered in it. Two paths conflict if they both require the usage of the same link. This conflict model is equivalent to an EP graph. Suppose we try to find a schedule for the messages such that no two messages sharing a link are scheduled at the same time interval. Then a proper vertex coloring of the EP graph corresponds to a feasible schedule on this network.

Often the host graphs are restricted to certain families such as paths, cycles, trees, grids, etc. When H is restricted to paths and cycles we get the well known families of interval graphs [11] and circular arc graphs [15], respectively. When H is restricted to trees, we obtain the family of Edge Intersection Graph of Paths in a tree (EPT) [6], and finally when H is a grid, the corresponding graph is termed an EPG graph [10].

In [2] we defined the graph of edge intersecting and non-splitting paths of a tree (ENPT) of a given representation $\langle T, \mathcal{P} \rangle$ as described above, denoted by $\textsc{Enpt}(\mathcal{P})$. This graph is a subgraph of $\textsc{Ept}(\mathcal{P})$: it has a vertex v for each path P_v of $\mathcal{P}$ and two vertices u, v of this graph are adjacent if the paths P_u and P_v edge-intersect and do not split (that is, their union is a path). A graph G is an ENPT graph if there is a tree T and a set of paths $\mathcal{P}$ of T such that $G = \textsc{Enpt}(\mathcal{P})$. We note that whenever T is a path $\textsc{Ept}(\mathcal{P}) = \textsc{Enpt}(\mathcal{P})$ is an Interval Graph. Therefore, the class ENPT includes all Interval Graphs. In this work we extend this definition to the case where the host graph is not necessarily a tree.

The motivation to study edge-intersecting and non-splitting paths arises from all-optical telecommunication networks. In the Wavelength Division Multiplexing (WDM) technology different signals can be multiplexed onto a single optical fiber by using different wavelength ranges of the laser beam [13]. A stream of signals traveling from its source to its destination in optical form is termed a *lightpath.* A lightpath is realized by signals traveling through a series of fibers, on a certain wavelength. Specifically, Wavelength Assignment problems (WLA) are a family of path coloring problems that aim to assign wavelengths (i.e. colors) to lightpaths, so that no two lightpaths with a common edge receive the same wavelength and certain objective function (depending on the problem) is minimized. *Traffic Grooming* is the term used for combination of several low-capacity requests into one lightpath using Time Division Multiplexing (TDM) technology [4]. In this context a set of paths can be combined into one lightpath, thus receiving the same color. One main constraint in traffic grooming is that the union of the paths receiving the same color should constitute a path or a set of disjoint paths, and this originated the study of the class of ENPT graphs.

1.2 Related Work

EPT graphs have been studied in the literature. Their recognition is NP-complete [5], whereas one can solve in polynomial time the maximum clique [5] and the maximum stable set [14] problems for this class.

Current research on intersection graphs is concentrated on the comparison of various intersection graphs of paths in a tree and their relation to chordal and weakly chordal graphs [7, 8]. The k-edge intersection graphs where two vertices are adjacent if their corresponding paths intersect on at least k edges are studied in [9].

Several recent papers consider the edge intersection graphs of paths on a grid. Since all graphs are EPG (see [10]), studies focus mostly on the sub-classes of EPG where the paths have a limited number of bends. An EPG graph is B_k-EPG if it admits a representation in which every path has at most k bends. The *bend number* of a graph G is the minimum number k such that G has a B_k-EPG representation. Clearly, a graph is B_0-EPG if and only if it is an interval graph. B_1-EPG graphs are studied in [10] in which every tree is shown to be B_1-EPG, and a characterization of C_4 representations is given. In [1] it is shown that there exists an outer-planar graph that is not a B_1-EPG. The recognition problem of B_1-EPG graphs is shown to be NP-complete in [12]. The work [1] investigates the bend number of some special graph classes.

In [2] we define the family of ENPT graphs and study their properties; in particular, we study the representations of induced cycles, that turns out to be much more complex than their counterpart in the EPT graphs (discussed in [6]).

In [3] we focus on graphs with bend number 1. We show that cycles and trees are B_1-ENPG graphs, that the recognition problem of these graphs in NP-complete in general, and provide a polynomial time recognition algorithm for Co-Bipartite graphs.

1.3 Our Contribution

In this work we extend the definition of the family of edge intersecting and non-splitting paths on a tree (ENPT) graphs to the general case in which the host graph is not necessarily a tree. In Section 3 we consider the general case and we show that not every co-bipartite graph is an ENP graph, and that the family of ENP graphs is exactly the family of ENPG graphs. Therefore, it is sufficient to consider ENPG graphs for most of the questions. We thus consider, in Section 4, the effect of restricting the number of bends of the paths, where a bend of a path on a grid is a pair of consecutive edges of the path one of which is vertical and the other is horizontal. We show that restricting the number of bends give rise to an infinite hierarchy of sub-families of ENPG graphs, where every family is properly included in the next family in the hierarchy.

2 Preliminaries

2.1 Graphs, Co-Bipartite Graphs

Given a simple graph (no loops or parallel edges) $G = (V(G), E(G))$ and a vertex v of G, we denote by $\delta_G(v)$ the set of edges of G incident to v, by $N_G(v)$ the set consisting of v and its neighbors in G, and by $d_G(v) = |\delta_G(v)|$ the degree of v in G. Whenever there is no ambiguity we omit the subscript G and write $d(v)$, $\delta(v)$ and $N(v)$. Two adjacent vertices u, v of G are *twins* if $N_G(u) = N_G(v)$. For a graph G, $\bar{V} \subseteq V(G)$ and $\bar{E} \subseteq E(G)$ we denote by $G[\bar{V}]$ and $G[\bar{E}]$ the subgraphs of G induced by $\bar{V}$ and by $\bar{E}$, respectively. The *union* of two graphs G, G' is the graph $G \cup G' \overset{def}{=} (V(G) \cup V(G'), E(G) \cup E(G'))$.

We emphasize that two graphs G and G' are equal if and only if $V(G) = V(G')$ and $E(G) = E(G')$. Therefore, two isomorphic graphs may be distinct, and they are considered as such in the counting arguments in our proofs.

A graph $G = (V, E)$ is *co-bipartite* if V can be partitioned into two cliques. Note that this partition is not necessarily unique. We denote bipartite and co-bipartite graphs as triples (V_1, V_2, E) where

a) $V_1 \cap V_2 = \emptyset$,
b) for bipartite graphs V_1 and V_2 are independent sets,
c) for co-bipartite graphs V_1 and V_2 are cliques,
d) $E \subseteq V_1 \times V_2$ (in other words E does not contain the cliques' edges).

Unless stated otherwise we assume that G is connected and none of V_1, V_2 is empty.

2.2 Walks, trails, paths, segments, splits

A *walk* in a graph $G = (V(G), E(G))$ is a sequence $P = (e_1, e_2, \ldots, e_\ell)$ of edges of $E(G)$ such that there are vertices $v_0, v_1, \ldots, v_\ell$ satisfying $e_i = \{v_{i-1}, v_i\}$ for every $i \in [\ell]$. Clearly, the reverse sequence $(e_\ell, \ldots, e_1)$ is also a walk. The *length* of P is the number ℓ of (not necessarily distinct) edges in the sequence. In this work we do not consider *trivial* (zero length) walks, as such walks do not intersect others. P is *closed* whenever $v_0 = v_\ell$, and *open* otherwise. A *trail* is a walk consisting of distinct edges. A *(simple) path* is a walk consisting of distinct vertices except possibly $v_0 = v_\ell$. A contiguous sub-sequence of a walk (resp. trail, path) is termed a sub-walk (resp. sub-trail, sub-path).

Let $P = (e_1, e_2, \ldots, e_\ell)$ be a trail with vertices $v_0, v_1, \ldots, v_\ell$ as above. For every $i \in [\ell - 1]$, the triple (e_i, v_i, e_{i+1}) is an *internal point* of P. Whenever P is closed, the triple $(e_\ell, v_\ell = v_0, e_1)$ too, is an internal point of P. We denote the set of internal points of P by $INT(P)$. We say that a vertex v is an internal vertex of P, or equivalently that P crosses v if v is in (i.e. is the second entry of) a triple in $INT(P)$. If P is open $END(P) \overset{def}{=} \{v_0, v_\ell\}$ and $TAIL(P) \overset{def}{=} \{(e_1, v_0), (e_\ell, v_\ell)\}$ are the sets of *endpoints* of P and tails of P, respectively. Given a set $\mathcal{P}$ of trails, we define $TAIL(\mathcal{P}) \overset{def}{=} \cup_{P \in \mathcal{P}} TAIL(P)$, $END(\mathcal{P}) \overset{def}{=} \cup_{P \in \mathcal{P}} END(P)$ and

$INT(\mathcal{P}) \overset{def}{=} \cup_{P \in \mathcal{P}} INT(P)$. For brevity, in the text we often refer to internal points as vertices and to tails as edges. Moreover, when we apply the intersection and union operations on two trails we consider them as sets of internal points and endpoints.

Given two trails $P = (e_1, e_2, \ldots, e_\ell)$ and $P' = (e'_1, e'_2, \ldots, e'_{\ell'})$, a *segment* of $P \cap P'$ is a maximal trail that constitutes a sub-trail of both P and P'. Since P and P' are trails, $P \cap P'$ is the union of edge disjoint segments. We denote this set by $\mathcal{S}(P, P')$. A tail (resp. endpoint) of a segment is *terminating* if it is in $TAIL(P, P')$ (resp. $END(P, P')$). A *split* of P and P' is a pair of internal points $(e_i, v_i, e_{i+1}), (e'_j, v'_j, e'_{j+1}) \in INT(P) \times INT(P')$ such that $v_i = v'_j$ and $\left|\{e_i, e_{i+1}\} \cap \{e'_j, e'_{j+1}\}\right| = 1$. Note that the common edge and the common vertex constitute a non-terminating tail of a segment of $P \cap P'$ and conversely every non-terminating tail of a segment corresponds to a split. We denote by $split(P, P')$ the set of all splits of P and P', which corresponds to the set of all non-terminating tails of the segments $\mathcal{S}(P, P')$.

$P \parallel P'$ denotes that P and P' are non-intersecting, i.e. *edge-disjoint*. Whenever P and P' intersect and $split(P, P') = \emptyset$ we say that P and P' are *non-splitting* and denote this by $P \sim P'$. Note that in this case all the endpoints of the segments are terminating. As there are at most 4 such endpoints $1 \leq |\mathcal{S}(P, P')| \leq 2$. Moreover, in this case $P \cup P'$ is a trail. When $split(P, P') \neq \emptyset$ we say that P and P' are *splitting* and denote this by $P \nsim P'$. Clearly, for any two paths P and P' exactly one of the following holds: $P \parallel P'$, $P \sim P'$, $P \nsim P'$. See Figure 1 for an example.

A *bend* of a trail P in a grid H is an internal point of P whose edges have different directions, i.e. one vertical and one horizontal.

2.3 The EP and ENP graph families

Let $\mathcal{P}$ be a set of trails in a graph H. The graphs $\textsc{Ep}(\mathcal{P})$ and $\textsc{Enp}(\mathcal{P})$ are such that $V(\textsc{Enp}(\mathcal{P})) = V(\textsc{Ep}(\mathcal{P})) = V$ and there is a one to one correspondence between $\mathcal{P}$ and V, i.e. $\mathcal{P} = \{P_v : v \in V\}$. Given two trails $P_u, P_v \in \mathcal{P}$, $\{u, v\}$ is an edge of $\textsc{Ep}(\mathcal{P})$ if and only if P_u and P_v have a common edge. Moreover, $\{u, v\}$ is an edge of $\textsc{Enp}(\mathcal{P})$ if and only if $P_u \sim P_v$. Clearly, $E(\textsc{Enp}(\mathcal{P})) \subseteq E(\textsc{Ep}(\mathcal{P}))$. A graph G is ENP if there is a graph H and a set of paths $\mathcal{P}$ of H such that $G = \textsc{Enp}(\mathcal{P})$. In this case $\langle H, \mathcal{P} \rangle$ is an ENP *representation* of G. Two representations are *equivalent* if they are representations of the same graph.

Let $\langle H, \mathcal{P} \rangle$ be a representation of an ENP graph G. $\mathcal{P}_e \overset{def}{=} \{P \in \mathcal{P} | \ e \in P\}$ denotes the set of trails of $\mathcal{P}$ containing the edge e of H. For a subset $S \subseteq V(G)$ we define $\mathcal{P}_S \overset{def}{=} \{P_v \in \mathcal{P} : v \in S\}$. When H is a tree (resp. grid) $\textsc{Ep}(\mathcal{P})$ is an EPT (resp. EPG) graph, and $\textsc{Enp}(\mathcal{P})$ is an ENPT (resp. ENPG) graph; these graphs are denoted also as $\textsc{Ept}(\mathcal{P})$, $\textsc{Epg}(\mathcal{P})$, $\textsc{Enpt}(\mathcal{P})$ and $\textsc{Enpg}(\mathcal{P})$. Unless otherwise stated, without loss of generality we assume that H is a complete graph, as any non-edge of H can be substituted with an edge without affecting $\textsc{Enp}(\mathcal{P})$.

Lemma 1. *Let K be a clique of an* ENP *graph. Then one of the following holds:*

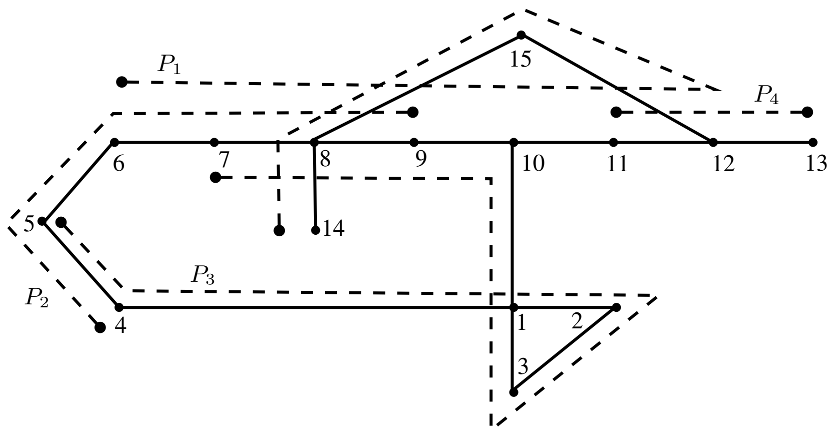


Fig. 1: The pairs of paths (P_2, P_4) and (P_3, P_4) do not share a common edge, therefore $P_2 \parallel P_4$ and $P_3 \parallel P_4$. P_1 and P_4 have a common edge $\{11, 12\}$, and 12 is a common internal vertex constituting a split of P_1 and P_4, therefore $P_1 \nsim P_4$. Similarly P_1 and P_3 have three common edges and 10 is a split vertex of P_1 and P_3, therefore $P_1 \nsim P_3$. P_1 and P_2 have three common edges but no splits, then $P_1 \sim P_2$. The same holds for the pair (P_2, P_3). However, we note that in the latter case the common edges are separated into two segments. The vertex 8 is not a split point of P_1 and P_2 because the only internal points of P_1 involving this vertex are $(e_{7,8}, 8, e_{7,9}), (e_{14,8}, 8, e_{8,15})$, and the only internal point of P_2 involving it is $(e_{7,8}, 8, e_{7,9})$. Moreover, $|\{e_{7,8}, e_{7,9}\} \cap \{e_{7,8}, e_{7,9}\}| = 2 \neq 1$ and $|\{e_{7,8}, e_{7,9}\} \cap \{e_{14,8}, e_{7,15}\}| = 0 \neq 1$.

i) $\cup \mathcal{P}_K$ is an open trail and $\cap P_K \neq \emptyset$.
ii) $\cup \mathcal{P}_K$ is a closed trail, and for every edge e of $\cup \mathcal{P}_K$ there exists an edge e' of $\cup \mathcal{P}_K$ such that $P \cap \{e, e'\} \neq \emptyset$ for every path $P \in \mathcal{P}_K$.

Proof. If $\cup \mathcal{P}_K$ contains two internal points (e_1, v, e_2) and (e'_1, v, e'_2) such that $|\{e_1, e_2\} \cap \{e'_1, e'_2\}| = 1$. Then there are two paths $P, P' \in \mathcal{P}_K$ such that $(e_1, v, e_2) \in INT(P)$ and $(e'_1, v, e'_2) \in INT(P')$. Therefore $split(P, P') \neq \emptyset$ and $P \not\sim P'$ contradicting the fact that K is a clique. Therefore $\cup \mathcal{P}_K$ s a disjoint union of trails. However, if $\cup \mathcal{P}_K$ contains two disjoint trails, then $P \parallel P'$ for any two paths P, P' from two distinct trails of $\cup pp_K$, contradicting the fact that K is a clique. Therefore $\cup \mathcal{P}_K$ is one trail.

i) If $\cup \mathcal{P}_K$ is an open trail, then we can embed it on the real line, so that the individual paths of $\mathcal{P}_K$ are intervals on the real line. Then, the result follows from the Helly property of intervals.
ii) If $\cup \mathcal{P}_K$ is an open trail, let e be any edge of this trail. Let $\mathcal{P}_e$ be the set of trails in $\mathcal{P}_K$ containing the edge e. Then $\cup(\mathcal{P}_K \setminus \mathcal{P}_e)$ is an open trail. By the previous result there is an edge e' of this trail that is contained of all these paths. Therefore, all the paths of $\mathcal{P}_K$ contain either e or e'.

□

Based on this lemma we say that K is an *open* (resp. *closed*) clique if $\cup \mathcal{P}_K$ is an open (resp. closed) trail. It will be convenient to use the following corollary of Lemma 1 in order to unify the two cases into one.

Corollary 1. *Let K be a clique of an* ENP *graph, with a representation $\langle H, \mathcal{P} \rangle$. Then $\cup \mathcal{P}_K$ is a sub-trail of a closed trail in which for every edge e there exists an edge e' such that $P \cap \{e, e'\} \neq \emptyset$ for every $P \in \mathcal{P}_K$.*

We denote a closed trail whose existence is guaranteed by Corollary 1 as $\mathcal{P}^{(K)}$. Note that $\mathcal{P}^{(K)}$ consists of at most one edge more than $\cup \mathcal{P}_K$

3 ENP

In this section we show that a) the family of ENP graphs does not include all co-bipartite graphs(Theorem 1), and b) the family of ENP graphs coincides with the family of ENPG graphs (Theorem 2).

We proceed with definitions regarding the relationship between the representations of two cliques. Given two vertex disjoint cliques K, K' of an ENP graph G with a representation $\langle H, \mathcal{P} \rangle$, we denote $\mathcal{S}(K, K') \overset{def}{=} \mathcal{S}(\mathcal{P}^{(K)}, \mathcal{P}^{(K')})$. A segment $S \in \mathcal{S}(K, K')$ is *quiet in K* if does not contain tails of paths of $\mathcal{P}_K$, and *busy in K*, otherwise. The importance of segments stems from the following observation:

Observation 31 *Consider a pair of trails $(P, P') \in \mathcal{P}_K \times \mathcal{P}_{K'}$. Then,*

i) $P \cap P' \subseteq \cup \mathcal{S}(K, K')$, and *ii) split$(P, P')$ corresponds to the set of all non-terminating segment endpoints crossed by both P and P'.*

$\mathcal{C}_{n,n}$ is the set of all co-bipartite graphs $G(K, K', E)$ where $K = [n]$ and $K' = \{i' : i \in [n]\}$. The following lemma bounds the number of graphs of this form as a function of the number of segments.

Lemma 2. *For any $s \geq 0$, the number of graphs $G = (K, K', E) \in \mathcal{C}_{n,n}$ with a representation $\langle H, \mathcal{P} \rangle$ such that $|\mathcal{S}(K, K')| \leq s$ is at most $(4n)!((2n + 2s)!)^2$.*

Theorem 1. Co-Bipartite $\not\subseteq$ ENP.

Proof. $|\mathcal{C}_{n,n}| = 2^{n^2}$ because there are n^2 pairs of vertices $(v, v') \in K \times K'$, and for every such pair, either $(v, v') \in E$ or $(v, v') \notin E$. In the rest of the proof we show that every $G \in \mathcal{C}_{n,n}$ has a representation $\langle H', \mathcal{P}' \rangle$ for which $s = |\mathcal{S}(K, K')| \leq 12n$. By Lemma 2, the number of such representations and therefore $|\mathcal{C}_{n,n} \cap \text{ENP}|$ is at most $(4n)!((2n + 2s)!)^2 \leq (4n)!((2n + 24n)!)^2 = (4n)!(26n)!(26n)!$. Therefore, $\log |\mathcal{C}_{n,n} \cap \text{ENP}| = O(n \log n)$, whereas $\log |\mathcal{C}_{n,n}| = n^2$ concluding the proof. It remains to show that G has a representation with $s \leq 12n$ segments.

The number of busy segments of $\mathcal{S}(K, K')$ is at most $4n$, because $|\mathit{END}(\mathcal{P}_K)| = 2n$ and an endpoint can be in at most 2 segments. We now bound the number of quiet segments of $\mathcal{S}(K, K')$. Consider two endpoints from $\mathit{END}(\mathcal{P}_K)$ that are consecutive on $\mathcal{P}^{(K)}$ and let P be the sub-trail of $\mathcal{P}^{(K)}$ between these two endpoints. By this choice, every trail of $\mathcal{P}_K$ intersecting P includes P. Let $\bar{\mathcal{S}}$ be the set of segments S that are sub-trails of P (thus $V(S) \subseteq \mathit{INT}(P)$). Suppose that $\left|\bar{\mathcal{S}}\right| > 4$. Consider the two edges $e_{a'}$ and $e_{b'}$ of $\mathcal{P}^{(K')}$ whose existence are guaranteed by Corollary 1. These two edges divide $\mathcal{P}^{(K')}$ into at most two open trails. One of these open trails contains (at least) 3 segments $S_1, S_2, S_3 \in \bar{\mathcal{S}}$ where the indices are in the order they appear on this open trail from $e_{a'}$ to $e_{b'}$ (see Figure 2). Let also v_{i1}, v_{i2} be the endpoints of S_i in the same order. We claim that the representation obtained by adding to H a new vertex x and two edges $\{v_{21}, x\}, \{x, v_{22}\}$ and finally modifying all the trails intersecting P (that therefore include S_2) so that the segment S_2 is replaced by the trail $(\{v_{21}, x\}, \{x, v_{22}\})$ is an equivalent representation. Clearly, any trail that does not intersect S_2 is not affected by this modification. Consider two trails P_v and $P_{v'}$ such that $(v, v') \in K \times K'$ and both intersect S_2. P_v includes P and therefore includes all the vertices of S_2, in particular crosses v_{21} and v_{22}. On the other hand, by Corollary 1, $P_{v'}$ contains at least one of $e_{a'}$ and $e_{b'}$. Without loss of generality let $e_{b'} \in P_{v'}$. Then, v_{22} is an internal vertex of $P_{v'}$. We conclude that $v_{22} \in \mathit{split}(P_v, P_{v'})$, i.e. $(v, v') \notin E(G)$. After the modification, we have $v_{31} \in \mathit{split}(P_v, P_{v'})$, thus (v, v') is not an edge of the resulting graph. Therefore, the new representation is equivalent to $\langle H, \mathcal{P} \rangle$. After this modification, S is not a segment of $\mathcal{S}(K, K')$ and the new representation has one segment less. We can apply this transformation until we get an equivalent representation $\langle H', \mathcal{P}' \rangle$ having at most 4 quiet segments between every two consecutive vertices of $\mathit{END}(\mathcal{P}_K)$. In other words, $\langle H', \mathcal{P}' \rangle$ has at most $8n$ quiet segments of $\mathcal{S}(K, K')$. Adding the at most $4n$ busy segments, we conclude that $s \leq 12n$. □

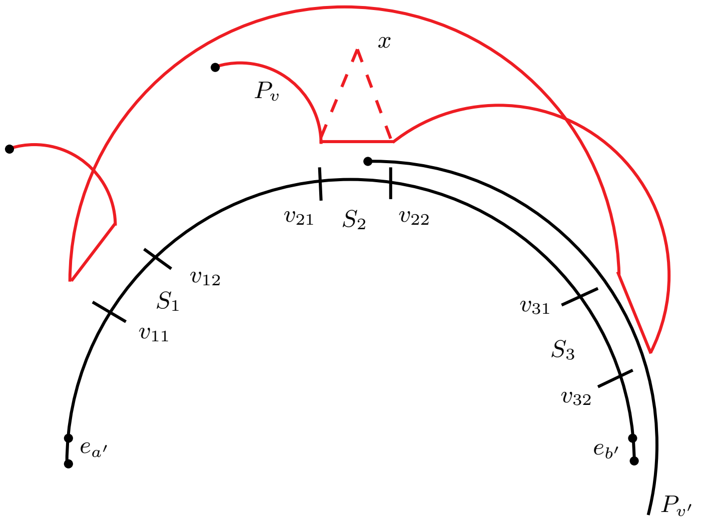


Fig. 2: Getting a representation with at most $8n$ quiet segments in the proof of Theorem 1. Whenever there are 3 segments on one side of the closed trail, the middle one can be bypassed.

Theorem 2. ENP=ENPG

Proof. Clearly, ENPG $\subseteq$ ENP. To prove the other direction, consider an ENP graph G with a representation $\langle H, \mathcal{P} \rangle$. We transform this representation into an equivalent ENPG representation, in three steps. In the first step, we obtain an equivalent representation $\langle H', \mathcal{P}' \rangle$ where H' is planar. In the second step, we transform $\langle H', \mathcal{P}' \rangle$ to an equivalent representation $\langle H'', \mathcal{P}'' \rangle$ where H'' is planar and $\Delta(H'') \leq 4$. Finally, we transform $\langle H'', \mathcal{P}'' \rangle$ to an ENPG representation.

The host graph H can be embedded in a plane such that the vertices are mapped to a set of points in general position on the plane and the edges are drawn as straight line segments. Specifically, no three points are co-linear and no three segments intersect at one point. Note that the mapping of the edges might intersect, however as the points are in general position, we can assume that no three edges intersect at the same point. For every intersection point of two edges e, e', we can add a vertex v to H and subdivide the edges e and e' (and consequently the paths in $\mathcal{P}$ containing e and e') such that the resulting 4 edges are incident to v. Every pair of paths P, P' that include e and e' respectively now intersect at v. However as we are not concerned with vertex intersections, the resulting representation is a representation of G. We continue in this way until all intersection points are replaced by a vertex. The graph H' of the resulting representation $\langle H', \mathcal{P}' \rangle$ is clearly planar.

We now transform the representation $\langle H', \mathcal{P}' \rangle$ to a representation $\langle H'', \mathcal{P}'' \rangle$ where H'' is planar with maximum degree at most 4. We start with $\langle H'', \mathcal{P}'' \rangle = \langle H', \mathcal{P}' \rangle$, and as long as there is a vertex v with $d_{H''}(v) > 4$, we eliminate such a vertex without introducing new vertices of degree more than 4 using the following procedure described in Figure 3: we number the edges incident to v as $e_1, e_2, \ldots, e_{d_v}$ in counterclockwise order according to the planar embedding of H'. Then e_1 and e_{d_v} are in the same face F of H''. We replace the vertex v with a path of d_v vertices $v_1, v_2, \ldots, v_{d_v}$ such that each edges e_i is incident to v_i. Clearly, the constructed path is part of F. We now construct the gadget in Figure 3 within the face F, where every path crossing v from an edge e_i to another edge e_j with $i < j$ is modified as described in the figure. Clearly, we do not lose intersections in this process. On the other hand, every pair of paths that intersect within the gadget have at least one edge incident to v in common before the transformation. Moreover, two paths have a split vertex within the gadget if and only if they split at v before the transformation.

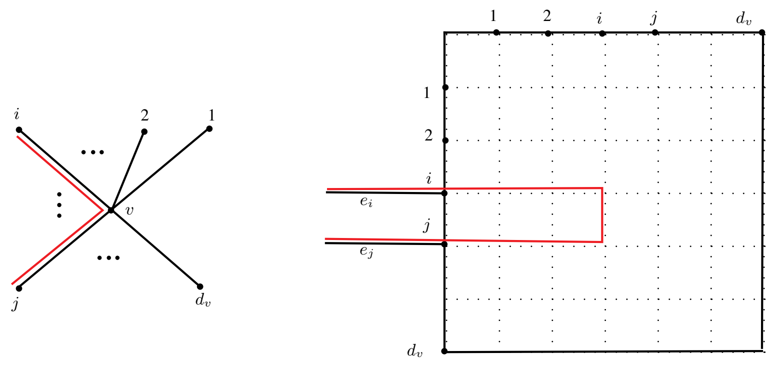


Fig. 3: The gadget used in the second transformation in the proof of Theorem 2

The last step is implied by the following theorem.

Theorem 2.3 [16]: A planar graph H'' with maximum degree at most 4 can be embedded in a grid graph H''' of polynomial size: the vertices u'' of H'' are mapped to vertices u''' of H'''; each edge $e'' = \{u'', v''\}$ of H'' is mapped to a path e''' between u''' and v''' in H'''; the intermediate vertices of e''' belong to exactly one such path. Given an embedding of H'' guaranteed by the theorem, we embed every trail $P'' \in \mathcal{P}''$ to a trail P''' of H''' by embedding every edge e'' of it to the corresponding path e''' of H'''. P''' is clearly a walk. P''' a trail, because otherwise there is an edge of H''' that is contained in the embedding of two distinct edges of H'', contradicting the last guarantee of the theorem. Clearly two trails P''_1, P''_2 of $\mathcal{P}''$ intersect if and only if the corresponding paths P'''_1, P'''_2 in

$\mathcal{P}'''$ intersect. Moreover a split $(e''_{11}, v'', e''_{12}), (e''_{21}, v'', e''_{22})$ of two paths P''_1, P''_2 is mapped to a split $(e'''_{11}, v''', e'''_{12}), (e''_{21}, v''', e''_{22})$ of the corresponding paths P'''_1, P'''_2 and this mapping is one to one. □

4 $\mathbf{B}_k$-ENPG

In this section we investigate the effect of restricting the number of bends of the paths on the family of ENPG graphs. An ENPG graph is B_k-ENPG if it has an ENPG representation $\langle H, \mathcal{P} \rangle$, in which every path $P \in \mathcal{P}$ has at most k bends. The graph $\mathrm{PM}_n \in \mathcal{C}_{n,n}$ is the co-bipartite graph (V, V', E) with $|V| = |V'| = n$ and E constitutes a perfect matching. In what follows we present upper and lower bounds for the bend number of this graph depending on n (lemmata 3 and 6). Using these bounds we show in Theorem 3 that for some infinite and increasing sequence of numbers $k_1, k_2, \ldots$ there is a graph in $\mathrm{B}_{k_{i+1}}$-ENPG that is not B_{k_i}-ENPG, thus proving the existence of an infinite hierarchy within the family of ENPG graphs. The following upper bound whose proof can be found in the appendix is implied by the representation in Figure 4.

Lemma 3. $\mathrm{PM}_n \in \mathrm{B}_{(2(n-1))}$-ENPG.

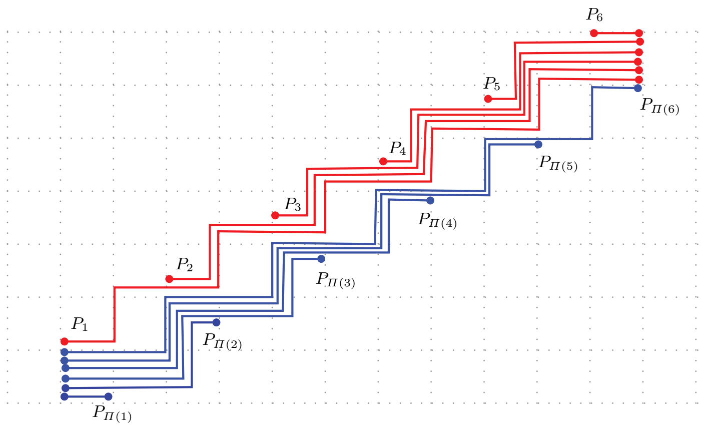


Fig. 4: The ENPG representation of the graph PM_n with $2(n-1)$ bends (proof of Lemma 3)

To prove the lower bound, we consider a given number k of bends and we upper bound the number n such that $\mathrm{PM}_n \in \mathrm{B}_k$-ENPG. We first show that a bound on the number of bends implies a bound on the total number of bends in the representation of a clique (Lemma 4). Then we show that this implies an upper bound on the number of segments (Lemma 5), and finally using this bound we bound n from above (Lemma 6).

Lemma 4. *Let K be a complete graph with* B_k-ENPG *representation* $\langle H, \mathcal{P} \rangle$. *Then* $\cup \mathcal{P}_K$ *contains at most* $2 \left\lfloor \frac{(3-\delta_K)k}{2} \right\rfloor$ *bends, where* $\delta_K = 1$ *when* K *is an open clique and* 0 *otherwise.*

Proof. If K is an open clique ($\delta_K = 1$), then there exists an edge e contained in every $P \in \mathcal{P}_K$. e divides $\cup \mathcal{P}_K$ into two trails each of which contains at most k bends. Therefore, $\cup \mathcal{P}_K$ contains at most $2k = 2 \left\lfloor \frac{(3-\delta_K)k}{2} \right\rfloor$ bends. If K is a closed clique ($\delta_K = 0$), consider a trail $P = (e_1, \ldots, e_\ell)$ of $\mathcal{P}_K$. Let P_1 (resp. P_ℓ) be a trail containing e_1 (resp. e_ℓ) with the maximum number of edges from $\cup \mathcal{P}_K \setminus P$. $P \cup P_1 \cup P_\ell = \cup \mathcal{P}_K$, because otherwise there is an edge $e \in \cup \mathcal{P}_K \setminus \{P, P_1, P_\ell\}$ that is included in some trail P' that does not contain neither e_1 nor e_ℓ, i.e. does not intersect P, contradicting the fact that K is a clique. We conclude that the number of bends of $\cup \mathcal{P}_K$ is at most $3k$. Moreover, this number is even because $\cup \mathcal{P}_K$ is a closed trail. Therefore the number of bends of $\cup \mathcal{P}_K$ is at most $2 \left\lfloor \frac{3k}{2} \right\rfloor = 2 \left\lfloor \frac{(3-\delta_K)k}{2} \right\rfloor$. □

Corollary 2. *A clique K of a* B_1-ENPG *graph is an open clique and* $\cup \mathcal{P}_K$ *has at most* 2 *bends.*

In the appendix we prove the following lemma

Lemma 5. *Let* $G = (K, K', E) \in \mathcal{C}_{n,n}$ *with a* B_k-ENPG *representation* $\langle H, \mathcal{P} \rangle$. *Then*

i) $|\mathcal{S}(K, K')| \leq 3k$.
ii) Moreover, this upper bound is asymptotically tight: there exist infinitely many pairs k, n *with a graph* $G \in \mathcal{C}_{n,n} \cap \mathrm{B}_k$-ENPG *such that* $|\mathcal{S}(K, K')| = 3k - 6$.

Lemma 6. *If* $n > 36k^2 + 6k$, *then* $\mathrm{PM}_n \notin \mathrm{B}_k$-ENPG.

Sketch of Proof: Let $\mathrm{PM}_n = (K, K', E)$ and assume by contradiction that PM_n is a B_k-ENPG graph with a corresponding ENPG representation $\langle H, \mathcal{P} \rangle$. Let $\mathcal{S} = \mathcal{S}(K, K')$. By Lemma 5, $|\mathcal{S}| \leq 3k$. Therefore, $|\mathcal{S}| (4 \cdot |\mathcal{S}| + 2) \leq 36k^2 + 6k < n$, i.e. $\frac{n}{|\mathcal{S}|} > 4 |\mathcal{S}| + 2$.

For an edge $e = (v, v') \in E$ we say that e is *realized* in segment $S \in \mathcal{S}$ if $P_v \cap P_{v'} \cap S \neq \emptyset$. Every edge $e = (v, v')$ is realized in at least one segment, because otherwise $P_v \cap P_{v'} = \emptyset$, contradicting the fact that $(v, v') \in E$. As $|E| = n$, there is at least one segment S in which $\ell \geq n / |\mathcal{S}| > 4 \cdot |\mathcal{S}| + 2$ edges are realized. As E is a perfect matching, these ℓ edges $E_S \subseteq E$ constitute a perfect matching of the subgraph $G_S = (K_S, K'_S, E_S) = \mathrm{PM}_\ell$ induced by all the endpoints of E_S.

We first show that there is at most one path of $\mathcal{P}_{K_S}$ crossing both endpoints of S. Assume by contradiction that there are two paths $P_u, P_v \in \mathcal{P}_{K_S}$ crossing both endpoints of S. Let u' be the unique neighbor of u in K'_S. Then $P_{u'} \subseteq S$ because otherwise $P_{u'}$ splits from P_u. Therefore, $P_v \supseteq S \supseteq P_{u'}$, i.e. $P_v \sim P_{u'}$ and $(v, u') \in E_S$, contradicting the fact that $G_S = \mathrm{PM}_\ell$. Similarly, there is at most one path from $\mathcal{P}_{K'_S}$ crossing both endpoints of S. We conclude that G_S

contains a subgraph $\bar{G}_S = (\bar{K}_S, \bar{K}'_S, \bar{E}_S) = \textsc{Pm}_{\bar{\ell}}$ where $\bar{\ell} \geq \ell - 2 > 4 \cdot |\mathcal{S}|$ with a representation in which every path crosses at most one endpoint of S.

Consider an edge $e = (v, v')$ of $\bar{G}_S$. P_v and $P_{v'}$ do not cross the same endpoint of S, as otherwise they would split. Let w, w' be the endpoints of S. There is a set $\bar{\bar{E}}_S$ of $\bar{\bar{\ell}} \geq \bar{\ell}/2 > 2 \cdot |\mathcal{S}|$ edges such that their endpoints induce a graph $\bar{\bar{G}}_S = (\bar{\bar{K}}_S, \bar{\bar{K}}'_S, \bar{\bar{E}}_S) = \textsc{Pm}_{\bar{\bar{\ell}}}$, where no path $P_v \in \mathcal{P}_{\bar{\bar{K}}_S}$ crosses w' and no path $P_{v'} \in \mathcal{P}_{\bar{\bar{K}}'_S}$ crosses w. All the paths of $\mathcal{P}_{\bar{\bar{K}}_S}$ (resp. $\mathcal{P}_{\bar{\bar{K}}'_S}$) have one endpoint in $\mathit{INT}(S)$. The other endpoint of such a path can be in one of the $|S|$ segments, or between two of them. We classify the paths of $\mathcal{P}_{\bar{\bar{K}}_S}$ by the location of the other endpoint. As $\bar{\bar{\ell}} > 2 \cdot |S|$, there are at least two paths P_{v_1} and P_{v_2} in the same category, i.e. crossing exactly the same segment endpoints. For $i \in \{1, 2\}$ let $P_{v'_i}$ be the unique neighbour of v_i in $\bar{\bar{K}}'_S$. $C = (v_1, v_2, v'_2, v'_1)$ is a cycle of $G = \textsc{Enpg}(\mathcal{P})$, therefore it is also a cycle of $\textsc{Epg}(\mathcal{P})$. We note that every pair of intersecting paths among these 4 paths intersect also in S. Then C is also a cycle of the interval graph obtained by restricting all the trails of $\langle H, \mathcal{P} \rangle$ to S. As an interval graph cannot contain an induced C_4, it must contain one chord. Assume without loss of generality that (v_1, v'_2) is such a chord. By definition, this chord is not in $\textsc{Pm}_n$. We conclude that $P_{v_1} \nsim P_{v'_2}$, implying $\mathit{split}(P_{v_1}, P_{v'_2}) \neq \emptyset$. As P_{v_1} and P_{v_2} cross exactly the same segment endpoints, we have $\mathit{split}(P_{v_2}, P_{v'_2}) = \mathit{split}(P_{v_1}, P_{v'_2}) \neq \emptyset$, contradicting the fact that (v_2, v'_2) is an edge of G. □

We are now ready to prove the main result of this section.

Theorem 3. *There is an infinite increasing sequence of integers* $\{k_i : i = 1, 2, \ldots\}$ *such that* B_{k_i}-ENPG $\subsetneq$ $\text{B}_{k_{i+1}}$-ENPG *for* $i \in \{1, 2, \ldots\}$.

Proof. Consider the family of B_k-ENPG graphs for some integer $k \geq 1$, and let $G = \textsc{Pm}_{36k^2+6k+1}$. $G \notin \text{B}_k$-ENPG by Lemma 6. On the other hand, by Lemma 3, $G \in \text{B}_{72k^2+12k}$-ENPG. Therefore B_k-ENPG $\subsetneq$ B_{72k^2+12k}-ENPG. We conclude that the infinite sequence defined as $k_{i+1} = 72k_i^2 + 12k_i$ for any $i > 1$ satisfies the claim where $k_1 \geq 0$ is chosen arbitrarily. □

5 Summary and Future Work

In this work we generalized the family of ENPT graphs to ENP graphs in which the host graphs can be any graph. We showed that without loss of generalization, we can restrict the host graphs to grids. Moreover, this family ENPG does not contain all graphs, as opposed to the family of EPG graphs that contains all graphs. We showed that by restricting the number of bends in the paths, we can define an infinite hierarchy of subfamilies of ENPG.

Although most of the results in this work are proved for cobipartite graphs, the counting arguments used in the proofs of Theorem 1 and Lemma 6 can be extended to the more general case of graphs with a large clique number. This extension is work in progress. Another open question is to find the sequence of the smallest numbers k_i such that B_{k_i}-ENPG $\subsetneq$ $\text{B}_{k_{i+1}}$-ENPG for every i. In particular, is B_1-ENPG $\subsetneq$ B_2-ENPG $\subsetneq \cdots$? In [2] we studied recognition problems in given graph pairs, i.e. a pair$((\textsc{Ept}(\mathcal{P}), \textsc{Enpt}(\mathcal{P}))$ of graphs that are defined on the same set of vertices. A similar study on EP, ENP graph pairs is another interesting research direction.

References

1. T. C. Biedl and M. Stern. On edge-intersection graphs of k-bend paths in grids. *Discrete Mathematics & Theoretical Computer Science*, 12(1):1–12, 2010.
2. A. Boyacı, T. Ekim, M. Shalom, and S. Zaks. Graphs of edge-intersecting non-splitting paths in a tree: Towards hole representations. In *The Proceedings of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Luebeck, Germany*, pages 115–126, June 2013.
3. A. Boyacı, T. Ekim, M. Shalom, and S. Zaks. Graphs of edge-intersecting and non-splitting one bend paths in a grid, 2014. In preparation.
4. O. Gerstel, R. Ramaswami, and G. Sasaki. Cost effective traffic grooming in wdm rings. In *INFOCOM'98, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 1998.
5. M. C. Golumbic and R. E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Mathematics*, 55(2):151 – 159, 1985.
6. M. C. Golumbic and R. E. Jamison. The edge intersection graphs of paths in a tree. *Journal of Combinatorial Theory, Series B*, 38(1):8 – 22, 1985.
7. M. C. Golumbic, M. Lipshteyn, and M. Stern. Equivalences and the complete hierarchy of intersection graphs of paths in a tree. *Discrete Appl. Math.*, 156:3203–3215, Oct. 2008.
8. M. C. Golumbic, M. Lipshteyn, and M. Stern. Representing edge intersection graphs of paths on degree 4 trees. *Discrete Mathematics*, 308(8):1381–1387, 2008.
9. M. C. Golumbic, M. Lipshteyn, and M. Stern. The k-edge intersection graphs of paths in a tree. *Discrete Appl. Math.*, 156:451–461, Feb. 2008.
10. M. C. Golumbic, M. Lipshteyn, and M. Stern. Edge intersection graphs of single bend paths on a grid. *Networks*, 54(3):130–138, 2009.
11. A. Hajnal and J. Surányi. Über die auflösung von graphen in vollständige teilgraphen. *Ann. Univ. Sci. Budapest Eötrös Sect. Math.*, 1:113–121, 1958.
12. D. Heldt, K. Knauer, and T. Ueckerdt. Edge-intersection graphs of grid paths: the bend-number. *Discrete Applied Mathematics*, 2013.
13. R. Ramaswami, K. N. Sivarajan, and G. H. Sasaki. *Optical Networks: A Practical Perspective.* Morgan Kaufmann Publisher Inc., San Francisco, 3rd edition, August 2009.
14. R. E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221 – 232, 1985.
15. A. Tucker. Characterizing circular-arc graphs. *Bulletin of the American Mathematical Society*, 76(6):1257–1260, 11 1970.
16. L. Yanpei, A. Morgana, and B. Simeone. General theoretical results on rectilinear embedability of graphs. *Acta Mathematicae Applicatae Sinica*, 7:187192, 1991.

A graph-easy class of mute lambda-terms

A. Bucciarelli[1], A. Carraro[2], G. Favro[1,2], and A. Salibra[2]

[1] Univ Paris Diderot, Sorbonne Paris Cité, PPS, UMR 7126, CNRS, Paris, France
[2] DAIS, Università Ca'Foscari Venezia, Italy

Abstract. Among the unsolvable terms of the lambda calculus, the *mute* (or *root-active*) ones are those having the highest degree of undefinedness. In this paper, we define an infinite set S of mute terms, and show that it is *graph-easy*: for any closed term t of the lambda calculus there exists a graph model equating all the terms of S to t.
Keywords: Lambda-calculus, mute terms, graph models, forcing, ultraproducts.

1 Introduction

It is a well known result by Jacopini [15] that Ω can be consistently equated to any closed term t of the (untyped) lambda-calculus, where Ω is the paradigmatic unsolvable term $(\lambda x.xx)(\lambda x.xx)$ (this is called *the easiness of* Ω). Baeten and Boerboom [3] gave the first semantic proof of this result by showing that for all closed terms t one can build a graph model satisfying the equation $\Omega = t$. This semantic result extends to other classes of models and to some other terms which share with Ω enough of its good will (cf. [7] for a survey of such results).

Mute lambda terms have been introduced by Berarducci [5], for defining models of the lambda calculus that do not identify all the unsolvable terms. Mute terms are somehow the "most undefined" lambda terms, as they are unsolvable of order 0 (zero terms), which are not β-convertible to a zero term applied to something else. For instance, Ω is mute, and $\Omega_3 = (\lambda x.xxx)(\lambda x.xxx)$ is a zero term that is not mute, since it reduces to $\Omega_3(\lambda x.xxx)$.

Berarducci proved that the set of mute terms is easy, in the sense that it is consistent with the lambda calculus to simultaneously equate all the mute terms to a fixed arbitrary closed term. Hereafter, a set of lambda terms that can be simultaneously, consistently equated to a fixed arbitrary closed term is called an *easy set*.

Given a class $\mathcal{C}$ of models of the lambda calculus, and an easy set S, we say that S is $\mathcal{C}$-easy if, for every closed term t, there exists a model in $\mathcal{C}$ that equates all the terms in S to t.

Studying $\mathcal{C}$-easiness gives insights on the expressive power of the class $\mathcal{C}$. Concerning filter lambda models, for instance, it had been conjectured [2] that they have full expressive power for singletons, in the sense that any easy singleton set is filter-easy. Carraro and Salibra [13] showed that this is not the case: there exists a co-r.e. set of easy terms that are not filter-easy. The first negative semantic result was obtained by Kerth [19]: $\Omega_3 I$, where $I = \lambda x.x$, is an easy term, but no graph model satisfies the identity $\Omega_3 I = I$. This result shows a limitation of graph models. The easiness of $\Omega_3 I$ was proven syntactically in [16] (see also [6]), but it was only given a semantic proof in [1], where the authors build, for each closed t, a filter model of $\Omega_3 I = t$.

Graph models are arguably the simplest models of the lambda calculus. There are two known methods for building graph models, namely: by *forcing* or by *canonical completion*. Both methods consist in completing a partial model into a total one.

The canonical completion method was introduced by Plotkin and Engeler and then systematized by Longo [21] for graph models. The word "canonical" refers here to the fact that the graph model is built inductively from the partial one and completely determined by it. This method was then used by Kerth [18] to prove the existence of 2^ω pairwise inconsistent graph theories, and by Bucciarelli–Salibra [11, 9, 10] to characterize minimal and maximal graph theories. In particular [11] shows that the minimal graph theory is not equal to the minimal lambda theory $\lambda\beta$, and that the lambda theory $\mathcal{B}$ (generated by equating lambda terms with the same Böhm tree) is the greatest sensible graph theory.

The forcing method originates with Baeten–Boerboom [3], and it is more flexible than canonical completions. In fact, the inductive construction depends here not only on the initial partial model but also on the consistency problem one is interested in. The method was afterwards generalized to other classes of webbed models by Jiang [17] and Kerth [20]. It was also generalized to families of terms similar to Ω by Zylberajch [23] and Berline–Salibra [8].

One more difference between these methods is that if we start with a recursive partial web, the canonical completion builds a recursive total web, while forcing always generates a non recursive web.

In this paper we define an infinite and recursive set of mute terms, the *regular mute* terms. A regular mute term has the form $s_0 s_1 \dots s_n$, for some n, and it has the property that, in n steps of head reduction, it reduces to a term of the same shape $t_0 t_1 \dots t_n$, where $t_0 = s_i$ for some $1 \leq i \leq n$. As regular mute terms are mute, we know that the set of all regular mute terms is easy, since each subset of an easy set is itself easy. We show that it is actually graph-easy by generalizing the forcing technique used in [8].

More precisely, given a closed λ-term t and a finite set $\{n_1, \dots, n_k\}$ of natural numbers, we construct a graph model which equates to t all the regular mute terms of the form $s_0 s_1 \dots s_{n_j}$, $1 \leq j \leq k$, using forcing.

Then we glue together these graph models in an ultraproduct, using a technique introduced in [12]. This gives rise to a graph model that is an expansion of the ultraproduct, where all the regular mute terms are equated to t, thus concluding the proof that the set of regular mute terms is graph-easy.

2 Theories and models of λ-calculus

With regard to the lambda-calculus we follow the notation and terminology of [4]. By Λ and Λ^o, respectively, we indicate the set of λ-terms and of closed λ-

terms. We denote $\alpha\beta$-conversion by $\lambda\beta$. A λ-*theory* is a congruence on Λ (with respect to the operators of abstraction and application) which contains $\lambda\beta$. A λ-theory is *consistent* if it does not equate all λ-terms, *inconsistent* otherwise.

It took some time, after Scott gave his model construction, for consensus to arise on the general notion of a model of the λ-calculus. There are mainly two descriptions that one can give: the category-theoretical and the algebraic one. The categorical notion of model, that of reflexive object in a Cartesian closed category (ccc), is well-suited for constructing concrete models, while the algebraic one is rather used to understand global properties of models (constructions of new models out of existing ones, closure properties, etc.) and to obtain results about the structure of the lattice of λ-theories. The algebraic description of models of λ-calculus proposes two kinds of structures, viz. the λ-*algebras* and the λ-*models*, both based on the notion of *combinatory algebra*. We will focus on λ-models.

A *combinatory algebra* $\mathbf{A} = (A, \cdot, \mathbf{k}, \mathbf{s})$ is a structure with a binary operation called *application* and two distinguished elements $\mathbf{k}$ and $\mathbf{s}$ called *basic combinators*. The symbol "$\cdot$" is usually omitted from expressions and by convention application associates to the left, allowing to leave out superfluous parentheses. The class of combinatory algebras is axiomatized by the equations $\mathbf{k}xy = x$ and $\mathbf{s}xyz = xz(yz)$. A function $f : A \to A$ is *representable* in $\mathbf{A}$ if there exists an element $a \in A$ such that $f(b) = ab$ for all $b \in A$. For example, the identity function is represented by the combinator $\mathbf{i} = \mathbf{skk}$.

The axioms of an elementary subclass of combinatory algebras, called λ-*models*, were expressly chosen to make coherent the interpretation of the λ-terms (see Barendregt [4, Def. 5.2.7]). In addition to five axioms due to Curry (see [4, Thm. 5.2.5]), the *Meyer-Scott axiom* is the most important one in the definition of a λ-model. In the first-order language of combinatory algebras it is formulated as $\forall xy.(\forall z.\ xz = yz) \Rightarrow \boldsymbol{\varepsilon} x = \boldsymbol{\varepsilon} y$, where the combinator $\boldsymbol{\varepsilon} = \mathbf{s}(\mathbf{ki})$ is made into an inner choice operator. Indeed, given any a, the element $\boldsymbol{\varepsilon} a$ represents the same function as a; by the Meyer-Scott axiom, $\boldsymbol{\varepsilon} c = \boldsymbol{\varepsilon} d$ for all c, d representing the same function.

Given a set A, we denote by Env_A the set of A-environments, i.e., the functions from the set Var of λ-calculus variables to A. For every $x \in \mathrm{Var}$ and $a \in A$ we denote by $\rho[x := a]$ the environment ρ' which coincides with ρ everywhere except on x, where ρ' takes the value a.

Given a λ-model $\mathbf{A}$, the interpretation $|t|^{\mathbf{A}} : Env_A \to A$ of a λ-term is defined by induction on the complexity of t in such a way that

$$|x|^{\mathbf{A}}_{\rho} = \rho(x); \qquad |tu|^{\mathbf{A}}_{\rho} = |t|^{\mathbf{A}}_{\rho} |u|^{\mathbf{A}}_{\rho}; \qquad |\lambda x.t|^{\mathbf{A}}_{\rho} = \boldsymbol{\varepsilon} b$$

where b is any element satisfying $ba = |t|^{\mathbf{A}}_{\rho[x:=a]}$ for every $a \in A$.

It is important to stress that the class of λ-models is axiomatized by first-order axioms expressed in terms of Horn formulas, so that it is closed under direct products; it is not axiomatized by equations only, so that it is not closed neither under substructures nor under homomorphic images.

3 Graph models

The class of graph models belongs to Scott's continuous semantics. Graph models owe their name to the fact that continuous functions are encoded in them via (a sufficient fragment of) their graphs, namely their traces.

A *graph model* is a model of untyped λ-calculus, which is generated from a web in a way that will be recalled below. Historically, the first graph model was Plotkin and Scott's P_ω (see e.g. [4]), which is also known in the literature as "the graph model". The simplest graph model, $\mathcal{E}$, was introduced soon afterwards, and independently, by Engeler [14] and Plotkin [22]. More examples can be found in [7].

As a matter of notation, we denote by D^* the set of all finite subsets of a set D. Elements of D^* will be denoted by small roman letters $a, b, c, \dots$, while elements of D by greek letters $\alpha, \beta, \gamma, \dots$.

For short we will confuse the model and its web and so we define:

Definition 1. *A graph model is a pair (D, p), where D is an infinite set and $p : D^* \times D \to D$ is an injective total function.*

Such a pair will also be called a *total pair*. In the setting of graph models a *partial pair* (see [7]) is a pair (A, q) where A is any set and $q : A^* \times A \rightharpoonup A$ is a partial (possibly total) injection. Examples of partial pairs are: the empty pair $(\emptyset, \emptyset)$ and all the graph models.

If (D, p) is a partial pair, we write $a \to_p \alpha$ (or $a \to \alpha$ if p is evident from the context) for $p(a, \alpha)$. Moreover, $\beta \to \alpha$ means $\{\beta\} \to \alpha$. $a_1 \to a_2 \to \dots \to a_{n-1} \to a_n \to \alpha$ stands for $(a_1 \to (a_2 \to \dots (a_{n-1} \to (a_n \to \alpha)) \dots))$. If $\bar{a} = a_1, a_2, \dots, a_n$, then $\bar{a} \to \alpha$ stands for $(a_1 \to (a_2 \to \dots (a_{n-1} \to (a_n \to \alpha)) \dots))$.

A total pair (D, p) generates a *λ-model* of universe $\mathcal{P}(D)$, called *graph λ-model*. In particular $\mathcal{P}(D)$ is endowed with an application operator that makes it a λ-model. The interpretation $|t|^p : Env_{\mathcal{P}(D)} \to \mathcal{P}(D)$ of a λ-term t relative to (D, p) can be described inductively as follows (see Section 2):

- $|x|^p_\rho = \rho(x)$
- $|tu|^p_\rho = \{\alpha : (\exists a \subseteq |u|^p_\rho)\ \ a \to \alpha \in |t|^p_\rho\}$
- $|\lambda x.t|^p_\rho = \{\, a \to \alpha : \alpha \in |t|^p_{\rho[x:=a]}\}$

Since $|t|^p_\rho$ only depends on the value of ρ on the free variables of t, we only write $|t|^p$ if t is closed.

A graph model (D, p) *satisfies* $t = u$, written $(D, p) \models t = u$, if $|t|^p_\rho = |u|^p_\rho$ for all environments ρ. The λ-theory $Th(D, p)$ induced by (D, p) is defined as

$$Th(D, p) = \{t = u : t, u \in \Lambda \text{ and } |t|^p = |u|^p\}.$$

A λ-theory induced by a graph model will be called a *graph theory*.

4 The regular mute λ-terms

A first step towards the definition of regular mute terms are the hereditarily n-ary terms, defined below.

Definition 2. *Let $n > 0$ and $\bar{x} \equiv x_1, \dots x_k$ be distinct variables. The set of hereditarily n-ary λ-terms over $\bar{x}$, written $H_n[\bar{x}]$, is the smallest set of λ-terms containing $x_1, \dots, x_k$ and satisfying the following property, for all fresh distinct variables $\bar{y} \equiv y_1, \dots, y_n$ and all terms $t_1, \dots, t_n$:*

$$t_1, \dots, t_n \in H_n[\bar{x}, \bar{y}] \;\Rightarrow\; \lambda\bar{y}.y_i t_1 \dots t_n \in H_n[\bar{x}].$$

We write H_n for $H_n[\,]$.

Example 1. Some unary and binary hereditary λ-terms:

- $\lambda x.xx \in H_1$
- $\lambda y.yx \in H_1[x]$
- $\lambda x.x(\lambda y.yx) \in H_1$
- $\lambda zy.yzx \in H_2[x]$
- $\lambda xy.x(\lambda zy.yzx)y \in H_2$.

Given a natural number n and variables $\bar{x}$ we define inductively an increasing sequence of sets of λ-terms, starting at $H_n[\bar{x}]$:

Definition 3. *Let $\bar{x} \equiv x_1, \dots x_k$ and $\bar{y} \equiv y_1, \dots, y_n$ be distinct fresh variables.*

- $H_n^0[\bar{x}] = H_n[\bar{x}]$
- $H_n^{m+1}[\bar{x}] = \{s[\overline{u}/\overline{y}] : s \in H_n^m[\bar{x}, \bar{y}], \bar{u} \equiv u_1, \dots, u_n \in H_n^m[\bar{x}]\}$
- $S_n[\bar{x}] = \bigcup_m H_n^m[\bar{x}]$.

We write S_n for $S_n[\,]$. For $t \in S_n[\bar{x}]$, we denote by $rk(t)$ the smallest number such that $t \in H_n^{rk(t)}[\bar{x}]$.

Lemma 1. *If $\bar{y}$ is a sequence of n distinct variables, $s \in S_n[\bar{x}, \bar{y}]$ and $\bar{t} \equiv t_1, \dots, t_n \in S_n[\bar{x}]$, then $s[\bar{t}/\bar{y}] \in S_n[\bar{x}]$.*

Lemma 2. *Let t be a λ-term. Then $t \in H_n^m[\bar{x}]$ if, and only if, there exist*

- *$s \in H_n^0[\bar{x}, \bar{z}^1, \dots, \bar{z}^m]$,*
- *sequences $\bar{z}^i$ ($i = 1, \dots, m$) of n distinct variables,*
- *sequences $\bar{t}^i$ ($i = 1, \dots, m$) of n terms $\bar{t}^i \equiv t_1^i, \dots, t_n^i \in H_n^{m-i}[\bar{x}, \bar{z}^1, \dots, \bar{z}^{i-1}]$*

such that $t \equiv s[\overline{t^m}/\overline{z^m}] \cdots [\overline{t^1}/\overline{z^1}]$.

Proof. Just an unfolding of the previous definition.

Proposition 1. *For all $n > 0$, $s_0, \dots, s_n \in S_n$, there exist $r_0, \dots, r_n \in S_n$ and $i \leq n$ such that*

$$s_0 s_1 \dots s_n \to_\beta^n r_0 r_1 \dots r_n \text{ and } r_0 \equiv s_i$$

Proof. (1) $rk(s_0) = 0$.

Since $s_0 \in H_n$, then $s_0 \equiv \lambda y_1 \dots y_n.y_i r_1 \dots r_n$ with $r_1, \dots, r_n \in H_n[y_1, \dots, y_n]$. Hence $s_0 s_1 \dots s_n \to_\beta^n s_i r_1[\bar{s}/\bar{y}] \dots r_n[\bar{s}/\bar{y}]$. By Lemma 1 the term $r_i[\bar{s}/\bar{y}] \in S_n$, and we are done.

(2) $rk(s_0) = m > 0$.

By Lemma 2 there exists $u \in H_n[\bar{z}^1, \dots, \bar{z}^m]$ such that $s_0 \equiv u[\bar{t}^m/\bar{z}^m] \dots [\bar{t}^1/\bar{z}^1]$, for some terms $\bar{t}^i \in H_n^{m-i}[\bar{z}^1, \dots, \bar{z}^{i-1}]$, for $1 \le i \le m$. The term u cannot be a variable because of the rank of s_0. Then by definition $u \equiv \lambda\bar{y}.y_i u_1 \dots u_n$ with $u_i \in H_n[\bar{z}^1, \dots, \bar{z}^m, \bar{y}]$. Then

$$s_0 = \lambda\bar{y}.y_i(u_1[\bar{t}^m/\bar{z}^m] \dots [\bar{t}^1/\bar{z}^1]) \dots (u_n[\bar{t}^m/\bar{z}^m] \dots [\bar{t}^1/\bar{z}^1])$$

and, if $\bar{s} = s_1, \dots, s_n$

$$s_0 s_1 \dots s_n \to_\beta^n s_i(u_1[\bar{t}^m/\bar{z}^m] \dots [\bar{t}^1/\bar{z}^1][\bar{s}/\bar{y}]) \dots (u_n[\bar{t}^m/\bar{z}^m] \dots [\bar{t}^1/\bar{z}^1][\bar{s}/\bar{y}]).$$

Theorem 1. *For all $s_0, \dots, s_n \in S_n$, the term $s_0 s_1 \dots s_n$ is mute.*

Hereafter, a term $s_0 s_1 \dots s_n$ ($s_i \in S_n$) is called a *n-regular mute term*; $\mathcal{M}_n$ will denote the set of all n-regular mute terms.

Example 2. Some unary and binary regular mute terms:

– $(\lambda x.xx)(\lambda x.xx) \in \mathcal{M}_1$
– $(\lambda x.x(\lambda y.yx))(\lambda x.xx) \in \mathcal{M}_1$
– $AAA \in \mathcal{M}_2$, where $A := \lambda xy.x(\lambda zt.tzx)y$.

Example 3. Let $B := \lambda x.x(\lambda y.xy)$. Then BB is a mute term that is not regular:

$$BB = (\lambda x.x(\lambda y.xy))B \to_\beta B(\lambda y.By) \to_\beta BB$$

5 Forcing for regular mute terms

In this section we show that, given a closed λ-term t and a finite set $\{n_1, \dots, n_k\}$ of natural numbers, there exists a graph model which equates all the regular mute terms of the form $s_0 s_1 \dots s_{n_j}$, $1 \le j \le k$, to t, using forcing.

5.1 Some useful lemmas

Lemma 3. *Let (D, p) be a graph model, ρ be D-environment and $\bar{\beta} = \beta, \beta, \dots, \beta$ (n-times). If $\beta = \bar{\beta} \to \alpha$, $t \in S_n[\bar{x}]$ and $\beta \in \rho(x_i)$ $(i = 1, \dots, k)$ then $\beta \in |t|^p_\rho$.*

Proof. Base case: $t \in H_n[\bar{x}]$. Let $\bar{u} \in H_n[\bar{x}, \bar{y}]$ and $z \in \{\bar{x}, \bar{y}\}$ such that $t = \lambda\bar{y}.z\bar{u}$.

$$\beta = \bar{\beta} \to \alpha \in |\lambda\bar{y}.z\bar{u}|^p_\rho \iff \alpha \in |z\bar{u}|^p_{\rho[\bar{y}:=\bar{\beta}]}$$

Since $\beta \in \rho[\bar{y} := \bar{\beta}]$ and by induction hypothesis $\beta \in |u_i|^p_{\rho[\bar{y}:=\bar{\beta}]}$, then $\alpha \in |z\bar{u}|^p_{\rho[\bar{y}:=\bar{\beta}]}$.

Let $t \in H_n^{m+1}[\bar{x}]$. Then $t \equiv s[\overline{u}/\overline{y}]$, where $s \in H_n^m[\bar{x}, \bar{y}]$ and $\bar{u} \equiv u_1, \dots, u_n \in H_n^m[\bar{x}]$. By induction hypothesis we have $\beta \in |u_i|_\rho^p$. Since $|s[\overline{u}/\overline{y}]|_\rho^p = |s|^p_{\rho[\bar{y}:=|\bar{u}|_\rho^p]}$ and $\beta \in \rho[\bar{y} := |\bar{u}|_\rho^p](y_i)$, then by induction hypothesis $\beta \in |s|^p_{\rho[\bar{y}:=|\bar{u}|_\rho^p]}$ and we get the conclusion.

Lemma 4. *Let (D, p) be a graph model, $s_0^0 s_1^0 \dots s_n^0 \in \mathcal{M}_n$ ($s_i^0 \in S_n$) and $\gamma \in |s_0^0 s_1^0 \dots s_n^0|^p$. Then there exist a sequence $\beta_i \equiv a_1^i \to \dots \to a_n^i \to \gamma$ ($i \in \omega$) of elements of D and a sequence d_i ($i \in \omega$) of natural numbers $\leq n$ such that $\beta_{i+1} \in a_{d_i}^i$.*

Proof. By Proposition 1 there exists an infinite sequence of mute terms such that

$$s_0^0 s_1^0 \dots s_n^0 \;\to_\beta^n\; s_0^1 s_1^1 \dots s_n^1 \;\to_\beta^n\; \dots \;\to_\beta^n\; s_0^k s_1^k \dots s_n^k \;\to_\beta^n \dots$$

and $s_0^k \equiv s_{d_{k-1}}^{k-1}$ for some $1 \leq d_{k-1} \leq n$. The number d_{k-1} is the order of the head variable of the term s_0^{k-1}. By $\gamma \in |s_0^0 s_1^0 \dots s_n^0|^p$ there exists $a_1^0 \to \dots \to a_n^0 \to \gamma \in |s_0^0|^p$ such that $a_i^0 \subseteq |s_i^0|^p$. We define

$$\beta_0 = a_1^0 \to \dots \to a_n^0 \to \gamma.$$

Assume $\beta_k = a_1^k \to \dots \to a_n^k \to \gamma \in |s_0^k|^p$ and $a_j^k \subseteq |s_j^k|^p$ for every $j \leq n$.

Since $s_0^k = \lambda \bar{y}.y_{d_k} u_1 \dots u_n$ for some terms u_i and $\beta_k \in |s_0^k|^p$, then, if $\bar{a} = a_1^k, \dots, a_n^k$

$$\gamma \in a_{d_k}^k (u_1[\bar{a}/\bar{y}]) \dots (u_n[\bar{a}/\bar{y}]).$$

Then there exists $\beta_{k+1} = a_1^{k+1} \to \dots \to a_n^{k+1} \to \gamma \in a_{d_k}^k \subseteq |s_0^{k+1}|^p = |s_{d_k}^k|^p$ and $a_j^{k+1} \subseteq |s_j^{k+1}|^p$.

5.2 Forcing at work

We recall the notion of weakly continuous operator from [8].

Definition 4. *Let D be an infinite countable set. By $\mathcal{I}(D)$ we indicate the cpo of partial injections $q : D^* \times D \rightharpoonup D$, ordered by inclusion of their graphs.*

By a "*total p*" we will mean "an element of $\mathcal{I}(D)$ which is a total map" (equivalently: which is a maximal element of $\mathcal{I}(D)$). The domain and range of $q \in \mathcal{I}(D)$ are denoted by *dom*(q) and *rg*(q). We will also confuse the partial injections and their graphs.

Definition 5. *A function $F : \mathcal{I}(D) \to \mathcal{P}(D)$ is weakly continuous if it is monotone with respect to inclusion and if furthermore, for all total $p \in \mathcal{I}(D)$,*

$$F(p) = \bigcup_{q \subseteq_{\text{fin}} p} F(q).$$

Let $p \in \mathcal{I}(D)$. The universe $U(p)$ of p is defined as follows:

$$U(p) = \bigcup_{(a,\alpha)\in dom(p)} (a \cup \{\alpha, p(a,\alpha)\}).$$

If p is finite, then the universe of p is also finite.

Let $p \in \mathcal{I}(D)$ be finite, $\alpha \in D$, $\bar{\epsilon} \equiv \epsilon_1, \dots, \epsilon_k \in D \setminus U(p)$ and $k \in \mathbb{N}$. Then we denote $p_{\bar{\epsilon},\alpha}$ the extension of p such that

$$\epsilon_2 = \epsilon_1 \to \alpha; \qquad \epsilon_{j+1} = \epsilon_1 \to \epsilon_j \ (j = 2, \dots, k-1); \qquad \epsilon_1 = \epsilon_1 \to \epsilon_k.$$

Notice that

$$\epsilon_1 = \epsilon_1 \to \epsilon_1 \to \cdots \to \epsilon_1 \to \alpha \quad (k\text{-times})$$

and $p_{\bar{\epsilon},\alpha}$ is also finite.

Let $e \subseteq_{\text{fin}} \mathbb{N}$. We let $\mathcal{M}_e = \bigcup_{i\in e} \mathcal{M}_i$ the set of n-regular mute terms for $n \in e$.

The next theorem is the main technical tool for proving the easiness of the full set of n-regular mute terms. It generalizes [8, Thm. 11].

Theorem 2. *Let $F : \mathcal{I}(D) \to \mathcal{P}(D)$ be a weakly continuous function and let $e \subseteq_{\text{fin}} \mathbb{N}$. Then there exists a total $p_e : D^* \times D \to D$ such that $(D, p_e) \models t = F(p_e)$ for all terms $t \in \mathcal{M}_e$.*

Proof. We are going to build an increasing sequence of finite injective maps p_n, starting from $p_0 = \emptyset$, and a sequence of elements $\alpha_n \in D \cup \{*\}$, where $*$ is a new element, such that: $p_e =_{def} \cup p_n$ is a total injection, and $(D, p_e) \models t = A = F(p_e)$ for all $t \in \mathcal{M}_e$, where $A =_{def} \{\alpha_n : n \in \omega\} \cap D$.

We fix an enumeration of D and an enumeration of $D^* \times D$.

We start from $p_0 = \emptyset$.

Assume that p_n and $\alpha_0, \dots, \alpha_{n-1}$ have been built. We let

- $\alpha_n =$ First element of $F(p_n) \setminus \{\alpha_0, \dots, \alpha_{n-1}\}$ in the enumeration of D, if this set is non-empty, and $\alpha_n = *$ otherwise;
- $(b_n, \delta_n) =$ "the first element in $(D^* \times D) \setminus dom(p_n)$";
- $\gamma_n =$ "the first element in $D \setminus (U(p_n) \cup b_n \cup \{\delta_n\} \cup \{\alpha_0, \dots, \alpha_{n-1}, \alpha_n\})$".

Let $r = p_n \cup \{\gamma_n = b_n \to_r \delta_n\}$.

Case 1: $\alpha_n = *$. We let $p_{n+1} = r$.

Case 2: $\alpha_n \in D$.

Let $e = \{k_1, \dots, k_m\}$. We define $q_0 \subseteq q_1 \subseteq \cdots \subseteq q_m \in \mathcal{I}(D)$ as follows: $q_0 = r$ and $p_{n+1} = q_m$. Assume we have defined q_i. We define $q_{i+1} = (q_i)_{\bar{\epsilon}^{n,i},\alpha_n}$ (see above), where

$$\bar{\epsilon}^{n,i} \equiv \epsilon_1^{n,i}, \dots, \epsilon_{k_{i+1}}^{n,i} \in D \setminus (U(q_i) \cup \{\alpha_n\})$$

are distinct elements.

It is clear that p_n is a strictly increasing sequence of well-defined finite injective maps and that $p_e = \cup p_n$ is total.

It is also clear that each p_n (and p_e) is partitioned into two disjoint sets: $p_n = p_n^1 \cup p_n^2$, where $p_n^1 = \{b_i \to \delta_i = \gamma_i : 1 \le i \le n-1\}$ is called the gamma part of p_n and $p_n^2 = p_n \setminus p_n^1$ is called the epsilon part.

For every $\gamma \in D$, we define

$$deg(\gamma) = \begin{cases} 0 & \text{if } \gamma \notin rg(p_e) \\ min\{n : \gamma \in rg(p_n)\} & \text{if } \gamma \in rg(p_e) \end{cases}$$

Moreover, $deg(c) = max\{deg(x) : x \in c\}$ for every $c \subseteq_{\text{fin}} D$.

The following lemmas easily derive from the construction of p_e since $(rg(p_{n+1}) \setminus rg(p_n)) \cap U(p_n) = \emptyset$.

Lemma 5. *If $deg(a \to \alpha) = n$ and $\alpha \notin rg(p_n)$, then $\alpha \notin rg(p_e)$.*

Lemma 6. *(i) $deg(a \to \alpha) \ge deg(a), deg(\alpha)$.*
(ii) If $a \to \alpha$ is in the gamma part of p_e, then $deg(a \to \alpha) > deg(a), deg(\alpha)$.

Lemma 7. *If $\alpha_n \in rg(p_e)$ then $deg(\alpha_n) \le n$.*

Lemma 8. *There exists no cycle $\beta = c_1 \to c_2 \to \dots c_m \to \beta$.*

Proof. Consider a minimal cycle $\beta_i = c_i \to \beta_{i+1}$ $(1 \le i \le m-1)$ and $\beta_m = c_m \to \beta_1$. By Lemma 6 we have $deg(\beta_1) \ge deg(\beta_2) \ge \dots \ge deg(\beta_m) \ge deg(\beta_1)$. Let us set this common degree equal to $k+1$. If $\beta_1 = \gamma_k = b_k \to_{p_{k+1}} \delta_k$ then $\delta_k = \beta_2$ has degree $k+1$. This is not possible by Lemma 6(ii). If $\beta_1 = \epsilon_j^{k,i}$ then $\epsilon_j^{k,i} = c_1 \to c_2 \to \dots c_m \to \epsilon_j^{k,i}$. From this it follows that either α_k has degree $k+1$ (contradicting Lemma 7) or $\epsilon_j^{k,i} = \epsilon_{j-l}^{k,i}$ (contradicting that the epsilon elements are distinct) or $\epsilon_j^{k,i} = \alpha_k$ (contradicting the definition of epsilon elements). This concludes the proof of the lemma.

There remains to see that $(D, p_e) \models t = A = F(p_e)$ for every $t \in \mathcal{M}_e$.

$A \subseteq F(p_e)$: it follows from the definition of α_n and from the fact that $F(p_n) \subseteq F(p_e)$.

$F(p_e) \subseteq A$: suppose $\gamma \in F(p_e)$; then, since F is weakly continuous, $\gamma \in F(p_m)$ for some m (and for all the larger ones). If $\gamma \notin A$ then, for all $n \ge m$, $\alpha_n \in D$ has smaller rank than γ in the enumeration of D, contradicting the fact that there is only a finite number of such elements.

Let $m \in e$ and $t \equiv s_0 s_1 \dots s_m \in \mathcal{M}_m$.
$A \subseteq |t|^{p_e}$: Let $\alpha_n \neq *$. The condition $(D, p_e) \models \alpha_n \in |t|^{p_e}$ follows immediately from Lemma 3 and the fact that

$$\epsilon_1^{n,m} = \epsilon_1^{n,m} \to \epsilon_1^{n,m} \to \dots \to \epsilon_1^{n,m} \to \alpha_n \quad (m\text{-times}).$$

$|t|^{p_e} \subseteq A$: Assume by contraposition that $\gamma \in |t|^{p_e}$ and $\gamma \neq \alpha_n$ for every n. Then by Lemma 4 there exist a sequence $\beta_j \equiv a_1^j \to \dots \to a_m^j \to \gamma$ $(j \in \omega)$ of elements of D and a sequence d_j $(j \in \omega)$ of natural numbers $\le m$ satisfying the property $\beta_{j+1} \in a_{d_j}^j$.

By Lemma 6 and by $\beta_{j+1} \in a^j_{d_j}$ the sequence $deg(\beta_j)$ is an infinite decreasing sequence of natural numbers. Then there exists j such that $deg(\beta_{j+i}) = deg(\beta_j) = n$ for all $i \geq 0$. Since p_n is finite, it must exist $k \geq j$ and $l > 0$ such that $\beta_k = \beta_{k+l}$.

Moreover, $n = deg(\beta_k) \geq deg(a^k_{d_k} \to a^k_{d_k+1} \to \cdots \to \gamma) \geq deg(\beta_{k+1}) = n$ because $\beta_{k+1} \in a^k_{d_k}$. Then $deg(a^{k+i}_{d_{k+i}} \to a^k_{d_k+1} \to \dots \gamma) = n$ for every $i \leq l$. Since $a^{k+i}_{d_{k+i}}$ cannot be $\{\epsilon_1\}$ (otherwise $\beta_{k+i+1} = \epsilon_1$ and $\gamma = \alpha_n$) and there is exactly one pair (b_{n-1}, δ_{n-1}) such that $((b_{n-1}, \delta_{n-1}), \gamma_{n-1}) \in p^1_n \setminus p_{n-1}$, then

$$a^{k+i}_{d_{k+i}} \to (a^k_{d_{k+i}+1} \to \dots \gamma) = a^{k+j}_{d_{k+j}} \to (a^k_{d_{k+j}+1} \to \dots \gamma), \quad \text{for every } i, j \leq l.$$

This implies that $a^{k+i}_{d_{k+i}} = a^{k+j}_{d_{k+j}}$, etc. Since by Lemma 8 there are no cycles, then we get $\beta_k = \beta_{k+1} = \cdots = \beta_{k+l-1} = \beta_{k+l}$. It follows that $\beta_k \in a^k_{d_k}$. Since $a^k_{d_k} \to a^k_{d_k+1} \to \cdots \to \gamma$ belongs to the gamma part of p_e, this contradicts Lemma 6(ii).

Definition 6. *(Forcing) For a term M, a partial pair (D, q), a D-environment ρ and $\alpha \in D$, the abbreviation $q \Vdash_\rho \alpha \in M$ means that for all total injections $p \supseteq q$ we have that $(D, p) \models \alpha \in |M|^p_\rho$. Furthermore $q \Vdash_\rho X \subseteq M$ means that $q \Vdash_\rho \alpha \in M$ for all $\alpha \in X$.*

If M is closed we write $q \Vdash \alpha \in M$ for $q \Vdash_\rho \alpha \in M$.

Thus, for p is total, $p \Vdash \alpha \in M$ if and only if $\alpha \in |M|^p$.

Lemma 9. *For every term M and environment ρ the function $F_{M,\rho} : \mathcal{I}(D) \to \mathcal{P}(D)$ defined by $F_{M,\rho}(q) = \{\, \alpha \in D : q \Vdash_\rho \alpha \in M\}$ is weakly continuous, and we have $F_{M,\rho}(p) = |M|^p_\rho$ for each total p.*

Proof. The proof of the weak continuity of $F_{M,\rho}$ is a straightforward induction on the complexity of M. Let $p \in Q$ be total. We have to show that $F_{M,\rho}(p) = \bigcup_{q \subseteq_{\text{fin}} p} F_{M,\rho}(q) = |M|^p_\rho$.

If M is a variable x then $F_{x,\rho}(q) = \{\, \alpha \in D : q \Vdash \alpha \in \rho(x)\}$ is the constant function with value $\rho(x)$.

If $M = PQ$ and $\alpha \in |M|^p_\rho$, then there exists $a \subseteq |Q|^p_\rho$ such that $p(a, \alpha) \in |P|^p_\rho$. Choose such an a and let $\gamma = p(a, \alpha)$. By induction hypothesis there is a finite $q \subseteq p$ such that $q \Vdash_\rho a \subseteq Q$ and a finite $r \subseteq p$ such that $r \Vdash_\rho \gamma \in P$; then it is clear that $q \cup r \cup \{((a, \alpha), \gamma)\} \Vdash \alpha \in M$.

If $M = \lambda x.P$ and $\alpha \in |M|^p_\rho$ then there is a unique pair (b, β) such that $\alpha = p(b, \beta)$ and $\beta \in |P|^p_{\rho[x:=b]}$. By induction hypothesis there is a finite $q \subseteq p$ such that $q \Vdash_{\rho[x:=b]} \beta \in P$; then it is clear that $q \cup \{((b, \beta), \alpha)\} \Vdash_\rho \alpha \in M$.

Theorem 3. *Let M be a closed term. Then, for every $e \subseteq_{\text{fin}} \omega$ there exists a graph model (D, p_e) such that $(D, p_e) \models t = M$ for all regular mute terms $t \in \mathcal{M}_e$.*

Proof. It is sufficient to consider an arbitrary environment ρ, the weakly continuous map $F_{M,\rho} : \mathcal{I}(D) \to \mathcal{P}(D)$ defined in Lemma 9 and the graph model (D, p_e) defined in Theorem 2.

6 Ultraproducts

Ultraproducts result from a suitable combination of the direct product and quotient constructions. They were introduced in the 1950's by Loś.

Let I be a non-empty set and let $\{\mathbf{A}_i\}_{i\in I}$ be a family of combinatory algebras. Let U be a proper ultrafilter of the boolean algebra $\mathcal{P}(I)$. The relation $\sim_U$, given by $a \sim_U b \iff \{i \in I : a(i) = b(i)\} \in U$, is a congruence on the combinatory algebra $\prod_{i\in I} \mathbf{A}_i$. The *ultraproduct* of the family $\{\mathbf{A}_i\}_{i\in I}$, noted $(\prod_{i\in I} \mathbf{A}_i)/U$, is defined as the quotient of the product $\prod_{i\in I} \mathbf{A}_i$ by the congruence $\sim_U$. If $a \in \prod_{i\in I} \mathbf{A}_i$, then we denote by a/U the equivalence class of a with respect to the congruence $\sim_U$. If all members of $\{\mathbf{A}_i\}_{i\in I}$ are λ-models, by a celebrated theorem of Loś we have that $(\prod_{i\in I} \mathbf{A}_i)/U$ is a λ-model too, because λ-models are axiomatized by first-order sentences. The basic combinators of the λ-model $(\prod_{i\in I} \mathbf{A}_i)/U$ are $\mathbf{k}/U$ and $\mathbf{s}/U$, and application is given by $x/U \cdot y/U = (x\cdot y)/U$, where the application $x \cdot y$ is defined pointwise.

We now recall the famous Loś theorem.

Theorem 4 (Loś). *Let $\mathcal{L}$ be a first-order language and $\{\mathbf{A}_i\}_{i\in I}$ be a family of $\mathcal{L}$-structures indexed by a non-empty set I an let U be a proper ultrafilter of $\mathcal{P}(I)$. Then for every $\mathcal{L}$-formula $\varphi(x_1, \dots, x_n)$ and for every tuple $(a_1, \dots, a_n) \in \prod_{i\in I} \mathbf{A}_i$ we have that*

$$(\prod_{i\in I} \mathbf{A}_i)/U \models \varphi(a_1/U, \dots, a_n/U) \iff \{i \in I : \mathbf{A}_i \models \varphi(a_1(i), \dots, a_n(i))\} \in U.$$

The following theorem is [12, Theorem 4.5].

Theorem 5. *Let $(D_j, p_j)_{j\in J}$ be a family of total pairs, $\mathbf{A} = (\mathbf{A}_j : j \in J)$ be the corresponding family of graph λ-models, where $\mathbf{A}_j = (\mathcal{P}(D_j), \cdot, \mathbf{k}, \mathbf{s})$, and let $\mathcal{F}$ be an ultrafilter on J. Then there exists a graph model (E, q) such that the ultraproduct $(\Pi_{j\in J}\mathbf{A}_j)/\mathcal{F}$ can be embedded into the graph λ-model determined by (E, q).*

Theorem 6. *Let M be a closed term and $\mathcal{M} = \bigcup_{n\in\mathbb{N}} \mathcal{M}_n$ be the set of all regular mute λ-terms. Then there exists a total pair (E, q) such that*

$$(E, q) \models M = t, \quad \text{for every } t \in \mathcal{M}.$$

Proof. Let
$$K := \{e \subseteq \mathbb{N} : e \text{ is finite}\}$$
and $\mathcal{F}$ be a non-principal ultrafilter on $\mathcal{P}(K)$ that contains the set

$$K_n = \{e : n \in e\}, \quad \text{for each } n \in \mathbb{N}.$$

Hence $\mathcal{F}$ contains

$$K_e = \{d : e \subseteq d\} \quad \text{for each } e \subseteq_{\text{fin}} \mathbb{N}.$$

For every $e \subseteq \mathbb{N}$, let (D, p_e) be the total pair determined by Theorem 3 and define $\mathbf{A}_e$ be the corresponding graph λ-model. We show that $(\Pi_{e \in K} \mathbf{A}_e)/\mathcal{F} \models M = t$ for every $t \in \mathcal{M}$. Let $t \in \mathcal{M}_n$. Since

$$K_n \subseteq \{e : \mathbf{A}_e \models M = t\}.$$

and $K_n \in \mathcal{F}$ then we have that $(\Pi_{e \in K} \mathbf{A}_e)/\mathcal{F} \models M = t$ and the conclusion is obtained.

References

1. Alessi, F., Dezani-Ciancaglini, M., Honsell, F.: Filter models and easy terms, Italian Conference on Theoretical Computer Science, LNCS 2202, Springer-Verlag, 17–37, 2001.
2. Alessi, F., Lusin, S.: Simple easy terms, in S. van Bakel (ed.), Intersection Types and Related Systems, ENTCS 70, Elsevier, 2002.
3. Baeten, J., Boerboom, B.: Omega can be anything it should not be, in Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, Serie A, Indag. Mathematicae 41, p.111–120, 1979.
4. Barendregt, H.P.: The lambda-calculus, its syntax and semantics, Studies in Logic vol. 103, North_Holland, revised edition 1984.
5. Berarducci, A.: Infinite λ-calculus and non-sensible models, in Logic and Algebra, eds. A. Ursini and P. Agliano, Lecture Notes in Pure and Applied Mathematics 180, Marcel Dekker Inc., 1996.
6. Berarducci, A., Intrigila, B.: Some new results on easy λ-terms, Theoretical Computer Science 121, 71–88, 1993.
7. Berline, C.: From Computation to foundations via functions and application: The lambda-calculus and its webbed models, Theor. Comput. Sci. 249, 81–161, 2000.
8. Berline, C., Salibra, A.: Easiness in graph models, Theoretical Computer Science 354(1), 4–23, 2006.
9. Bucciarelli, A., Salibra, A.: The minimal graph-model of lambda-calculus, 28th Internat. Symp. on Math. Foundations of Comput. Science, LNCS 2747, Springer-Verlag, 2003.
10. Bucciarelli, A., Salibra, A.: The sensible graph theories of lambda-calculus, LICS'04.
11. Bucciarelli, A., Salibra, A.: Graph lambda-theories, Mathematical Structures in Computer Science 18(5), 975–1004, 2008.
12. Bucciarelli, A., Carraro, A., Salibra, A.: Minimal lambda-theories by ultraproducts, EPTCS 113, 2012.
13. Carraro, A., Salibra, A.: Easy lambda-terms are not always simple, RAIRO - Theor. Inform. and Applic. 46(2), 291–314, 2012.
14. Engeler, E.: Algebras and combinators, Alg. Univ. 13(3), 289–371, 1981.
15. Jacopini, C.: A condition for identifying two elements in whatever model of combinatory logic, in C. Böhm, ed., LNCS 37, Springer Verlag, 1975.
16. Jacopini, C., Venturini-Zilli, M.: Easy terms in the lambda-calculus, Fundamenta Informaticae VIII.2, 225–233, 1985.
17. Jiang, Y.: Consistency of a λ-theory with n-tuples and easy terms, Archives of Math. Logic, 34(2), 79–96, 1995.

18. Kerth, R.: Isomorphism and equational equivalence of continuous λ-models, Studia Logica 61, 403–415, 1998.
19. Kerth, R.: Isomorphisme et équivalence équationnelle entre modèles du λ-calcul, Thèse, Université Paris 7, 1995.
20. Kerth, R.: Forcing in stable models of untyped λ-calculus, Indagationas Mathematicae 10 , 59–71, 1999.
21. Longo, G.: Set-theoretical models of λ-calculus : theories, expansions and isomorphisms, Annals of Pure and Applied Logic 24, 153–188, 1983.
22. Plotkin, G.: A set-theoretical definition of application, Memorandum MIP-R-95, School of artificial intelligence, University of Edinburgh, 1972.
23. Zylberajch, C.: Syntaxe et sémantique de la facilité en λ-calcul, Thèse, Université Paris 7, 1991.

Relating threshold tolerance graphs to other graph classes

Tiziana Calamoneri[1]* and Blerina Sinaimeri[2]

[1] Sapienza University of Rome
via Salaria 113, 00198 Roma, Italy.
[2] INRIA and Université de Lyon
Université Lyon 1, LBBE, CNRS UMR558, France.

Abstract. A graph $G = (V, E)$ is a *threshold tolerance* if it is possible to associate weights and tolerances with each node of G so that two nodes are adjacent exactly when the sum of their weights exceeds either one of their tolerances. Threshold tolerance graphs are a special case of the well-known class of tolerance graphs and generalize the class of threshold graphs which are also extensively studied in literature. In this note we relate the threshold tolerance graphs with other important graph classes. In particular we show that threshold tolerance graphs are a proper subclass of co-strongly chordal graphs and strictly include the class of co-interval graphs. To this purpose, we exploit the relation with another graph class, *min leaf power graphs (mLPGs)*.

Keywords: threshold tolerance graphs, strongly chordal graphs, leaf power graphs, min leaf power graphs.

1 Introduction

In the literature, there exist hundreds of graph classes (for an idea of the variety and the extent of this, see [2]), each one introduced for a different reason, so that some of them have been proven to be in fact the same class only in a second moment. It is the case of threshold graphs, that have been introduced many times with different names and different definitions (the interested reader can refer to [13]). Threshold graphs play an important role in graph theory and they model constraints in many combinatorial optimization problems [8, 12, 17]. In this paper we consider one of their generalizations, namely threshold tolerance graphs.

A graph $G = (V, E)$ is a *threshold tolerance* graph if it is possible to associate weights and tolerances with each node of G so that two nodes are adjacent exactly when the sum of their weights exceeds either of their tolerances. More formally, there are positive real-valued functions, weights g and tolerances t on V such that $\{x, y\} \in E$ if and only if $g(x) + g(y) \geq \min(t(x), t(y))$. In the following we denote by *TT* the class

* This work was supported in part by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project "AMANDA' - Algorithmics for MAssive and Networked DAta"

of threshold tolerance graphs and indicate with $G = (V, E, g, t)$ a graph in this class. Threshold tolerance graphs have been introduced in [14] as a generalization of threshold graphs (we refer to this class by *Thr*). Indeed, threshold graphs constitute a proper subclass, and can be obtained by defining the tolerance function as a constant [15]. Specifically, a graph $G = (V, E)$ is a *threshold* graph if there is a real number t and for every vertex v in V there is a real weight a_v such that: $\{v, w\}$ is an edge if and only if $a_v + a_w \geq t$ ([15, 13]).

A *chord* of a cycle is an edge between two non consecutive vertices x, y of the cycle. A chord between two vertices x, y in an even cycle C is odd, when the distance in C between x and y is odd. A graph is *chordal* if every cycle of length at least 4 has a chord.

A graph is *strongly chordal* if it is chordal and every cycle of even length at least 6 has an odd chord.

Strongly chordal graphs can be also characterized in terms of excluding subgraphs. A k-sun (also known as trampoline), for $k \geq 3$, is the graph on $2k$ vertices obtained from a clique $\{c_1, \dots, c_k\}$ on k vertices and an independent set $\{s_1, \dots, s_k\}$ on k vertices and edge $\{s_i, c_i\}, \{s_i, c_{i+1}\}$ for all $1 \leq i < k$, and $\{s_k, c_k\}, \{s_k, c_1\}$.

A graph is strongly chordal if and only if it does not contain either a cycle on at least 4 vertices or a k-sun as an induced subgraph [10]. Strongly chordal graphs are a widely studied class of graphs that is characterized by several equivalent definitions that the interested reader can find in [2]. We will call *SC* the class of strongly chordal graphs.

A graph is *co-strongly chordal* if its complement is a strongly chordal graph.

It is known [15] that every threshold tolerance graph is co-strongly chordal but it is not known whether there exist graphs that are co-strongly chordal but not threshold tolerance; in other words, it is not known whether the inclusion is strict or not. In fact, in ISGCI (Information System on Graph Classes ands their Inclusions) [9] it is conjectured that these two classes could be possibly equal.

We provide a graph that belongs to the class of co-strongly chordal graphs but not to the class of threshold tolerance graphs, so proving that these two classes do not coincide.

A graph is a *tolerance graph* [11] if to every node v can be assigned a closed interval I_v on the real line and a tolerance t_v such that x and y are adjacent if and only if $|I_x \cap I_y| \geq \min\{t_x, t_y\}$, where $|I|$ is the length of the interval I. We will call *Tol* the class of tolerance graphs.

A graph is an *interval graph* if it has an intersection model consisting of intervals on a straight line. Clearly interval graphs are included in tolerance graphs and can be obtained by fixing a constant tolerance function. We will call *Int* the class of interval graphs.

It is known that co-*Tol* includes *TT* and that *TT* includes co-*Int*; while it can be easily derived that co-*Tol* properly includes *TT*, it is not known whether the other inclusion is strict or not (again, in ISGCI [9] it is conjectured that *TT* could be possibly equal to co-*Int*). We prove that both the inclusions are proper.

2 Preliminaries

In order to prove that *TT* is properly included in co-*SC*, we need to introduce the classes of leaf power graphs (*LPG*) and min leaf power graphs (*mLPG*).

A graph $G(V, E)$ is a *leaf power graph* [16] if there exists a tree T, a positive edge weight function w on T and a nonnegative number d_{max} such that there is an edge $\{u, v\}$ in E if and only if for their corresponding leaves in T, l_u, l_v, we have $d_{T,w}(l_u, l_v) \leq d_{max}$, where $d_{T,w}(l_u, l_v)$ is defined as the sum of the weights of the edges of T on the (unique) path between l_u and l_v. In symbols, we will write $G = LPG(T, w, d_{max})$.

A *t-caterpillar* is a tree in which all the nodes are within distance 1 of a central path, called spine, constituted of t nodes.

Although there has been a lot of work on this class of graphs (for a survey on this topic see e.g. [1]), a complete description of leaf power graphs is still unknown.

The following result is particularly relevant for our reasoning.

Fact 1 *[1] LPG is a proper subclass of SC. Furthermore, the graph in Figure 1 is a strongly chordal graph and not a leaf power graph.*

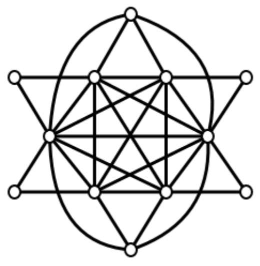


Fig. 1: A strongly chordal graph which is not in LPG [1].

The class mLPG is defined similarly to the class of leaf power graphs reversing the inequality in the definition. Formally, a graph $G = (V, E)$ is a *min leaf power graph (mLPG)* [4] if there exists a tree T, a positive edge weight function w on T and an integer d_{min} such that there is an edge (u, v) in E if and only if for their corresponding leaves in T l_u, l_v we have $d_{T,w}(l_u, l_v) \geq d_{min}$; in symbols, $G = mLPG(T, w, d_{min})$.

In [3] it is proved that $LPG \cap mLPG$ is not empty, and that neither of the classes LPG and mLPG is contained in the other. Furthermore, a number of papers deal with this class with a special focus on the intersection with LPGs (e.g. see [5–7]).

The next result will be useful in the following.

Fact 2 *[3] The class co-LPG coincides with mLPG and, vice-versa, the class co-mLPG coincides with LPG.*

The main issue related to LPG and mLPG is to prove that a certain class belongs to them by providing a constructive method that, given a graph, defines tree T, edge-weight function w and value d_{min} or d_{max}.

In the next section we will prove that threshold tolerance graphs are mLPGs and use this fact to separate the class TT from other graph classes.

3 Threshold tolerance graphs are mLPGs

Before proving the main result of this section, i.e. that threshold tolerance graphs are mLPGs, we need to demonstrate a preliminary lemma stating that, when we deal with threshold tolerance graphs, w.l.o.g. we can restrict ourselves to the case when g and t take only positive integer values.

Lemma 1. *A graph $G = (V, E)$ is a threshold tolerance if and only if there exist two functions $g, t : V \to N^+$ such that (V, E, g, t) is threshold tolerance.*

Proof. Clearly if f, g exist then by definition G is a threshold tolerance graph. Suppose now G is a threshold tolerance graph which weight and tolerance functions g and t are both defined from V to $\mathbb{R}^+$. We show that nevertheless, it is not restrictive to assume that $g, t : V \to \mathbb{Q}^+$ in view of the density of rational numbers among real numbers. So, we can assume that, for each $v \in V$, $t(v) = n_v/d_v$. Let m be the minimum common multiple of all the numbers d_v, $v \in V$. So we can express $t(v)$ as $t(v) = \frac{n_v \cdot m/d_v}{m}$ where m/d_v is an integer.

Define now the new functions g' and t' as $g'(v) = g(v) \cdot m$ and $t'(v) = t(v) \cdot m$, $v \in V$. Clearly, it holds that $g' : V \to \mathbb{Q}^+$ while $t' : V \to \mathbb{N}^+$.

In order to prove the claim, it remains to prove that g' and t' define the same graph defined by t and g. This descends from the fact that $g'(x) + g'(y) = (g(x) + g(y)) \cdot m \geq \min(t(x), t(y)) \cdot m = \min(t'(x), t'(y))$ if and only if $g(x) + g(y) \geq \min(t(x), t(y))$. $\square$

Theorem 1. *Threshold tolerance graphs are mLPGs.*

Proof. Let $G = (V, E, g, t)$ be a threshold tolerance graph. Let $K = \max_v t(v)$. In view of Lemma 1, it is not restrictive to assume that $g : V \to \mathbb{N}^+$, so we split the nodes of G in groups $S_1, \ldots, S_K$ such that $S_i = \{v \in V(G) : t(v) = i\}$. Observe that for some values of i the set S_i can be empty. We associate to G a caterpillar T as in Figure 2.

The spine of the caterpillar is formed by K nodes, $x_1, \ldots, x_K$, and each node x_i is connected to the leaves l_v corresponding to nodes v in S_i. The weights w of the edges of T are defined as follows:

- For each edge of the spine $w(x_i, x_{i+1}) = 0.5$ for $0 \leq i \leq K - 1$.
- For each leaf l_v connected to the spine through node x_i we assign a weight $w(v, x_i) = g(v) + \frac{K-t(v)}{2}$.

We show that $G = mLPG(T, w, K)$. To this purpose consider two nodes u and v in G. By construction, in T we have that l_u is connected to $x_{t(u)}$ and l_v to $x_{t(v)}$, where $t(u)$ and $t(v)$ are not necessary distinct. Clearly, w.l.o.g we can assume $t(v) \geq t(u)$, i.e. $t(u) = \min(t(u), t(v))$. We have that

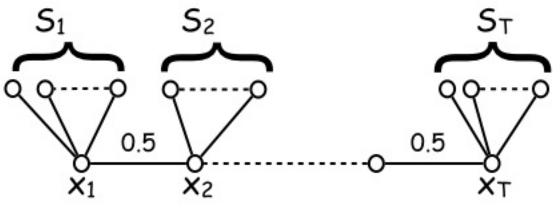


Fig. 2: The caterpillar used in the proof of Theorem 1 to prove that threshold tolerance graphs are in mLPG.

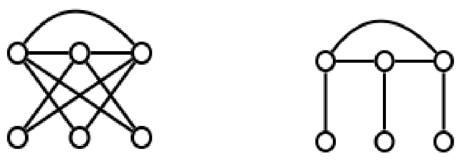
$$\begin{aligned} d_T(l_u, l_v) &= w(l_u, x_{t(u)}) + \frac{t(v) - t(u)}{2} + w(l_v, x_{t(v)}) \\ &= g(u) + \frac{K - t(u)}{2} + \frac{t(v) - t(u)}{2} + g(v) + \frac{K - t(v)}{2} \\ &= g(u) + g(v) + K - t(u) \end{aligned}$$

Clearly, $d_T(l_u, l_v) \geq K$ if and only if $g(u) + g(v) \geq t(u) = \min(t(u), t(v))$ and this proves the assertion. □

Now we are ready to prove the following Theorem.

Theorem 2. *$TT \subsetneq$ co-SC.*

Proof. Observe that according to the previous facts, TT are included in *mLPG* which in turn is strictly included in co-strongly chordal class of graphs. This proves the claim. □



(a) (b)

Fig. 3: (a) S_3 (b) $\bar{S_3}$

4 Separating threshold tolerance graphs from other graph classes

It is known [15] that co-$Tol \subseteq TT$. This inclusion is in fact proper; indeed, the sun of dimension 3, S_3, shown in Figure 3(a), is a tolerance graph but not a co-threshold tolerance graph [15]. It follows that its complement, $\bar{S_3}$, shown in Figure 3(b), is a co-tolerance graph but not a threshold tolerance graph, so proving co-$Tol \subset TT$.

It is easy to see that the graph S_3 belongs to the class of split antimatchings that are provably included in LPG but not in mLPG [4]. Furthermore, $\bar{S_3}$ is a co-threshold tolerance graph [15], i.e. S_3 is a threshold tolerance graph;finally, $\bar{S_3}$ is a split matching, and hence included in mLPG but not in LPG [4].

These inclusions prove the following:

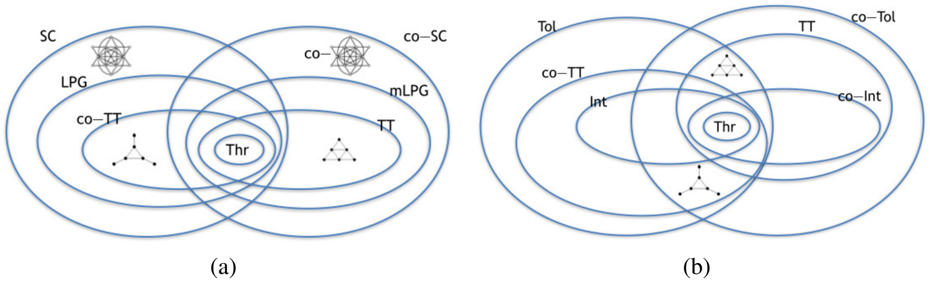


Fig. 4: Graphical representation of the known inclusions among the classes handled in this paper.

Theorem 3. *The classes $TT \setminus LPG$ and co-$TT \setminus$mLPG are not empty.*

Collecting these results and those reported in [4, 15] about S_3 and $\bar{S_3}$ we can finally conclude that $S_3 \in (Tol \cap TT) \setminus$ (co-$TT \cup$ co-Int) while $\bar{S_3} \in$ (co-$TT \cap$ co-$Tol) \setminus (TT \cup Int)$. The results obtained are depicted in Fig. 4.

5 Conclusions and Open Problems

In this paper, we clarified the relation between some classes of graphs. In particular, we have been able to position some special graphs, in order to prove that some class inclusions are strict. In particular, we proved that threshold tolerance graphs are strictly included in co-strongly chordal graphs, so confuting a conjecture reported in [9]. In order to do this, we exploit mLPGs deducing as a side effect that threshold tolerance graphs are mLPGs.

We summarize the obtained results in the two diagrams of Figure 4, from which it naturally arises an interesting open problem: how are related tolerance graphs and leaf power graphs (and, analogously, co-tolerance and min leaf power graphs)?

References

1. A. Brandstädt, On Leaf Powers, Technical report, University of Rostock, (2010).
2. A. Brandstädt, V. B. Le, J. Spinrad, Graph classes: a survey, SIAM Monographs on discrete mathematics and applications (1999).
3. T. Calamoneri, E. Montefusco, R. Petreschi, B. Sinaimeri, Exploring Pairwise Compatibility Graphs, Theoretical Computer Science , 468 (2013) 23–36.

4. T. Calamoneri, R. Petreschi, B. Sinaimeri, On relaxing the constraints in pairwise compatibility graphs, In: Md. S. Rahman and S.-i. Nakano (Eds.), WALCOM 2012, LNCS vol. 7157, Springer, Berlin (2012) 124–135.
5. T. Calamoneri, R. Petreschi, B. Sinaimeri, On the Pairwise Compatibility Property of Some Superclasses of Threshold Graphs, Discrete Mathematics, Algorithms and Applications, 5(2) (2013).
6. T. Calamoneri, R. Petreschi, On pairwise compatibility graphs having Dilworth number two, Theoretical Computer Science 524 (2014) 34–40.
7. T. Calamoneri, R. Petreschi, On Dilworth k Graphs and Their Pairwise Compatibility. WALCOM 2014, LNCS, Springer, Berlin (2014) 213–224.
8. V. Chvátal, P. L. Hammer, Set-packing and threshold graphs, Res.Report, Comp.Sci. Dept. Univ. of Waterloo, Ontario, (1973).
9. H.N. de Ridder et al. Information System on Graph Classes and their Inclusions (ISGCI), `http://www.graphclasses.org`.
10. M. Farber, Characterizations of strongly chordal graphs, Discrete Mathematics 43 (1983) 173–189.
11. M. C. Golumbic, C. L. Monma, W. T. Trotter Jr., Tolerance graphs, Discrete Applied Mathematics, 9(2), (1984) 157–170,
12. P. B. Henderson, Y. Zalcstein, A Graph-Theoretic Characterization of the PV-chunk Class of Synchronizing Primitives. SIAM J.Comput 6(1), (1977) 88–108.
13. N.V.R. Mahadev, U.N. Peled, Threshold Graphs and Related Topics, Elsevier, (1995).
14. C.L. Monma, B. Reed, W.T. Trotter Jr., A generalization of threshold graphs with Tolerance, Congressus Numerantium 55 (1986) 187–197.
15. C.L. Monma, B. Reed, W.T. Trotter Jr., Threshold Tolerance Graphs, J. Graph Theory 12 (1988) 343–362.
16. N. Nishimura, P. Ragde, D. M. Thilikos, On graph powers for leaf-labeled trees, J. Algorithms 42 (2002) 69–108.
17. E. T. Ordman, Minimal threshold separators and memory requirements for synchronization, SIAM Journal on Computing, Vol. 18, (1989) 152–165.

Černý-like problems for finite sets of words [⋆]

Arturo Carpi[1] and Flavio D'Alessandro[2]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia,
Via Vanvitelli 1, 06123 Perugia, Italy.
carpi@dmi.unipg.it

[2] Dipartimento di Matematica, Università di Roma "La Sapienza"
Piazzale Aldo Moro 2, 00185 Roma, Italy.
dalessan@mat.uniroma1.it

Abstract. This paper situates itself in the theory of variable length codes and of finite automata where the concepts of completeness and synchronization play a central role. In this theoretical setting, we investigate the problem of finding upper bounds to the minimal length of synchronizing and incompletable words of a finite language X in terms of the length of the words of X. This problem is related to two well-known conjectures formulated by Černý and Restivo respectively. In particular, if Restivo's conjecture is true, our main result provides a quadratic bound for the minimal length of a synchronizing pair of any finite synchronizing complete code with respect to the maximal length of its words.

Keywords: Černý conjecture, synchronizing automaton, incompletable word, synchronizing set, complete set

1 Introduction

The concepts of completeness and synchronization play a central role in Computer Science since they appear in the study of several problems on variable length codes and on finite automata. According to a well-known result of Schützenberger, the property of completeness provides an algebraic characterization of finite maximal codes, which are the objects used in Information Theory to construct optimal sequential codings. Let X be a set of words on an alphabet A. The set X is *complete* if any word on the alphabet A is a factor of some word belonging to X^*, otherwise it is *incomplete*. In the latter case, any word which is factor of no word of X^* is said to be *incompletable in* X. In [18], Restivo conjectured that a finite incomplete set X has always an incompletable word whose length is quadratically bounded by the maximal length of the words of X. Results on this problem have been obtained in [5, 14, 15, 18].

The property of synchronization plays a natural role in Information Theory where it is relevant for the construction of decoders that are able to efficiently cope with decoding errors caused by noise during the data transmission. A set

⋆ This work was partially supported by MIUR project PRIN 2010/2011 "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

X is *synchronizing* if there are two words u, v of X^* such that whenever $ruvs \in X^*$, $r, s \in A^*$, one has also $ru, vs \in X^*$. The pair of words (u, v) is called *synchronizing pair of X*.

In the study of synchronizing sets, the search for synchronizing words of minimal length in a prefix complete code is tightly related to that of synchronizing words of minimal length for synchronizing complete deterministic automata and the celebrated Černý Conjecture [12] (see also [1–3, 6–12, 16, 17, 20] for some results on the problem). In particular, in [2] (see also [3]), Béal and Perrin have proved that a complete synchronizing prefix code X on an alphabet of d letters with n code-words has a synchronizing word of length $O(n^2)$.

In this paper we are interested in finding upper bounds to the minimal lengths of incompletable and synchronizing words of a finite set X in terms of the size of X. We recall that the size of X is the parameter $\ell(X)$ defined as the maximal length of the words of X.

Let $\mathcal{L}$ be a class of finite languages. For all $n, d > 0$, we denote by $R_{\mathcal{L}}(n, d)$ the least positive integer r satisfying the following condition: any incomplete set $X \in \mathcal{L}$ on a d-letter alphabet such that $\ell(X) \leq n$ has an incompletable word of length r. Similarly, we denote by $C_{\mathcal{L}}(n, d)$ the least positive integer c satisfying the following condition: any synchronizing set $X \in \mathcal{L}$ on a d-letter alphabet such that $\ell(X) \leq n$ has a synchronizing pair (u, v) such that $|uv| \leq c$.

In this context, the main result of this paper provides a bridge between the parameters $R_{\mathcal{L}}(n, d)$ and $C_{\mathcal{L}}(n, d)$. More precisely, denoting by $\mathcal{F}$ and by $\mathcal{M}$ the classes of finite languages and of complete finite codes respectively, we show that, for all $n, d > 0$,

$$C_{\mathcal{M}}(n, d) \leq 2R_{\mathcal{F}}(n, d + 1) + 2n - 2.$$

In particular, if Restivo's conjecture is true, the latter bound gives

$$C_{\mathcal{M}}(n, d) = O(n^2),$$

thus providing a quadratic bound in the size of the set for the minimal length of a synchronizing pair of a finite synchronizing complete code.

In the second part of the paper, we study the dependence of the parameters $R_{\mathcal{L}}(n, d)$ and $C_{\mathcal{L}}(n, d)$ upon the number of letters d of the considered alphabet, by showing that both the parameters have a low rate of growth. More precisely, we show that, for the class $\mathcal{L}$ of finite languages (resp. codes, prefix codes), we have

$$R_{\mathcal{L}}(n, d) \leq \left\lceil \frac{R_{\mathcal{L}}(\lceil \log_2 d \rceil n, 2)}{\lfloor \log_2 d \rfloor} \right\rceil,$$

and, for the class $\mathcal{L}$ of finite complete languages (resp. codes, prefix codes), we have

$$C_{\mathcal{L}}(n, d) \leq \left\lceil \frac{C_{\mathcal{L}}(\lceil \log_2 (d + 1) \rceil n, 2)}{\lfloor \log_2 (d - 1) \rfloor} \right\rceil.$$

A similar result is obtained also when $\mathcal{L}$ is the class of finite (not necessarily complete) languages (resp. codes, prefix codes).

2 Preliminaries

In this section we shortly recall some basic results of the theory of automata and of the theory of codes which will be useful in the sequel and we fix the corresponding notation used in the paper. The reader can refer to [4, 13] for more details.

Let A be a finite alphabet and let A^* be the free monoid of words over the alphabet A. The identity of A^* is called the *empty word* and is denoted by ϵ. The *length* of a word $w \in A^*$ is the integer $|w|$ inductively defined by $|\epsilon| = 0$, $|wa| = |w| + 1$, $w \in A^*$, $a \in A$. Given $w \in A^*$ and $a \in A$, we denote by $|w|_a$ the number of occurrences of the letter a in w. For any finite set W of words we denote by $\ell(W)$ the maximal length of the words of W. The number $\ell(W)$ will be called the *size* of W. Given words $u, w \in A^*$, u is said to be a *factor* of w if $w = \alpha u \beta$, for some $\alpha, \beta \in A^*$. The set of all factors of w is denoted by $\mathrm{Fact}(w)$. Given a set W of words, the set of the factors of all the words of W is denoted by $\mathrm{Fact}(W)$. Similarly, given a word w, a word u is said to be a *prefix* of w if $w = u\beta$, for some $\beta \in A^*$. A set X is said to be *prefix* if, for any $u, v \in X$, with $v = uw$, with $w \in A^*$, one has $w = \epsilon$.

Definition 1. *Let X be a subset of A^*. A pair of words (r, s) is an* X-completion *of a word w if $rws \in X^*$. A word having an X-completion is a* completable *word of X; conversely, a word with no X-completion is an* incompletable *word of X. The set X is* complete *if all words of A^* are completable words of X; X is* incomplete*, otherwise.*

Another crucial notion of this paper is that of synchronizing set.

Definition 2. *A pair $(u, v) \in X^* \times X^*$ is a* synchronizing pair *of X if for every X-completion (r, s) of uv, it holds that*

$$ru, vs \in X^* .$$

The set X is synchronizing *if it has a synchronizing pair.*

The notion of synchronizing pair of a set is strictly related to that of *constant*. A word c of X^* is said to be a *constant* of X if, for every $u_1, u_2, u_3, u_4 \in A^*$ such that $u_1cu_2, u_3cu_4 \in X^*$, one has $u_1cu_4, u_3cu_2 \in X^*$. The following result holds.

Lemma 1. *Let X be a subset of A^*. If (u, v) is a synchronizing pair of X, uv is a constant of X. Conversely, if c is a constant of X, (c, c) is a synchronizing pair of X.*

2.1 Complete and synchronizing codes

The notions of complete and synchronizing sets provide a rich structure in the case that the set is a code. It is worth to shortly describe some fundamental results on such sets. A set X of words over an alphabet A is said to be a *(variable length) code over A* if it fulfills the unique factorization property, that is, for every word $u \in X^*$, there exists a unique sequence $x_1, \ldots, x_k$ of words of X such that $u = x_1 \cdots x_k$. A well-known example of codes is given by prefix sets. The notion of code is strictly related to the one of *monomorphism.* Let A and B be two alphabets. A map $h : A^* \to B^*$ is said to be a monomorphism or (sequential) encoding if h is injective and, for every $u, v \in A^*$, one has $h(uv) = h(u)h(v)$. The following lemma shows a well-known link between the notion of code and that of monomorphism.

Proposition 1. *Let $h : A^* \to B^*$ be a monomorphism. Then the set $X = h(A)$ is a code over B. If X is a code over B and X has the same cardinality of A, then every bijection between A and X can be extended to a (unique) monomorphism $h : A^* \to B^*$.*

In view of Proposition 1, the monomorphism defined by a prefix code will be called *prefix encoding.* A submonoid of A^* is said to be *free* if it is generated by a code. An important result due to Schützenberger provides a characterization of a free submonoid of A^* in terms of a property called *stability.* A submonoid N of A^* is said to be *stable* if, for every word $u \in A^*$, the existence of words n_1, n_2, n_3, n_4 of N such that $n_2 = un_1$ and $n_4 = n_3u$ implies $u \in N$.

Theorem 1. *Let N be a submonoid of A^*. Then N is a free submonoid of A^* if and only if N is stable.*

Let us finally recall some well-known results on codes.

Theorem 2. *(Kraft-McMillan inequality) Let A be a d-letter alphabet with $d \geq 2$ and let $k_1, \ldots, k_n$ be a finite sequence of positive integers such that*

$$\sum_{i=1}^{n} d^{-k_i} \leq 1 .$$

Then $k_1, \ldots, k_n$ is the sequence of the code-word lengths of a prefix code over A.

Given a code X over an alphabet A, X is said to be *maximal* if it is not properly contained in any other code over A. Let us recall another important result due to Schützenberger. It provides a tight relation between maximal and complete codes.

Theorem 3. *Let A be a d-letter alphabet and let X be a regular code of A^*. The following conditions are equivalent:*

- *X is a maximal code;*
- *X is a complete code;*
- $\sum_{x \in X} d^{-|x|} = 1$.

Given a set X over an alphabet A, X is said to be *maximal prefix* if it is prefix and it is not properly contained in any other prefix code over A. In the case of prefix codes we get:

Theorem 4. *Let X be a prefix code of A^* and suppose that X is a regular language. The following conditions are equivalent:*

- *X is maximal prefix;*
- *X is right complete, that is, for every word $u \in A^*, uA^* \cap XA^* \neq \emptyset$;*
- *X is a maximal code.*

It is worth recalling that Theorems 3 and 4 hold in the more general case of *thin codes* that is, for all the subsets X of A^* such that $X \cap A^*wA^* = \emptyset$, for some word $w \in A^*$. Another relevant result concerning synchronizing and complete codes is the following.

Theorem 5. *Let X be a synchronizing and complete code. Then there exists a constant $c \in X^*$ such that $cA^*c \subseteq X^*$. Conversely, if X is a code such that there exists a word $c \in X^*$ with $cA^*c \subseteq X^*$, then X is synchronizing and complete.*

2.2 Synchronizing automata and the Černý conjecture

A finite non-deterministic automaton is a tuple $\mathcal{A} = \langle Q, A, \delta, I, F\rangle$ where Q is a finite set of elements called *states*, δ is a map

$$\delta : Q \times A \longrightarrow \mathcal{P}(Q),$$

where $\mathcal{P}(Q)$ is the power set of Q, and I, F are subsets of Q. The map δ is called the *transition function* of $\mathcal{A}$ while I is called the *set of the initial states* and F is the *set of the final states.* The canonical extension of the map δ to the set $Q \times A^*$ is still denoted by δ. If P is a subset of Q and u is a word of A^*, we denote by $\delta(P, u)$ and $\delta(P, u^{-1})$ the sets:

$$\delta(P, u) = \{\delta(s, u) \mid s \in P\}, \quad \delta(P, u^{-1}) = \{s \in Q \mid \delta(s, u) \in P\}.$$

If no ambiguity arises, the sets $\delta(P, u)$ and $\delta(P, u^{-1})$ are denoted Pu and Pu^{-1}, respectively. With the automaton $\mathcal{A}$, we can associate a directed labelled multigraph $G = (Q, E)$, where the set E of edges is defined as $E = \{(p, a, q) \in Q \times A \times Q \mid \ q \in \delta(p, a)\}$. We recall that if $p, q \in Q$, then $q \in pu$ with $u \in A^*$, is equivalent to the existence of a path $c = p \xrightarrow{u} q$ in G labelled by u. The label of c is denoted by $||c||$. A word $u \in A^*$ is said to be accepted by $\mathcal{A}$ if $Iu \cap F \neq \emptyset$. The language accepted by $\mathcal{A}$, denoted by $L_{\mathcal{A}}$, is the set of all the words accepted by $\mathcal{A}$. An automaton $\mathcal{A}$ is said to be *unambiguous* if the following condition holds: for every $q_1, q_2, q_3, q_4 \in Q$ and for every $u, v \in A^*$ one has

$$q_2, q_3 \in q_1 u, \quad q_4 \in q_2 v \cap q_3 v \implies q_2 = q_3.$$

An automaton $\mathcal{A}$ is said to be *deterministic* if for every $q \in Q$ and for every $a \in A$, $\mathrm{Card}(qa) \leq 1$.

An automaton $\mathcal{A}$ is said to be *complete* if for every $u \in A^*$, there exists some $q \in Q$ such that $\mathrm{Card}(qu) \geq 1$.

An automaton $\mathcal{A}$ is said to be *transitive* if for every $p, q \in Q$, there exists $u \in A^*$ such that $q \in pu$. In the sequel, we will only consider automata $\mathcal{A}$ such that $I = F = \{1\}$, that is, with a unique initial and final state denoted 1. In particular, the tuple of the automaton $\mathcal{A}$ will be simply denoted $\mathcal{A} = \langle Q, A, \delta, 1\rangle$. Moreover, in the sequel, we will only consider transitive automata.

An unambiguous automaton $\mathcal{A}$ is said to be *synchronizing* if there exist two words $w_1, w_2 \in A^*$ such that $Qw_1 \cap Qw_2^{-1} = \{1\}$.

As is well known, a deterministic automaton $\mathcal{A}$ is synchronizing if and only if there is a word u such that $\mathrm{Card}(Qu) = 1$. Such a word is said to be a *synchronizing word* of $\mathcal{A}$. The following celebrated conjecture has been raised in [12].

Černý Conjecture. *Each synchronizing and complete deterministic automaton with n states has a synchronizing word of length $(n-1)^2$.*

Let us recall an important problem related to the Černý Conjecture. Let G be a finite, directed multigraph with all its vertices of the same outdegree. Then G is said to be *aperiodic* if the greatest common divisor of the lengths of all cycles of the graph is 1. The graph G is called a *Road coloring graph* (*RC-graph* for short) if it is aperiodic and strongly connected. A *synchronizing coloring* of G is a labeling of the edges of G that transforms it into a complete, deterministic and synchronizing automaton. The *Road coloring problem* asks for the existence of a synchronizing coloring for every RC-graph. In 2007, Trahtman proved the following remarkable result [19].

Theorem 6. *Every RC-graph has a synchronizing coloring.*

Let us conclude this section by recalling some well-known properties of the automaton that accepts the submonoid generated by a finite set (see, e.g., [6]). Let X be a finite set of words over an alphabet A. By using a standard construction, one can associate with X a transitive automaton denoted by $\mathcal{A}_X = \langle Q, A, \delta, 1\rangle$ that accepts X^*.

Lemma 2. *Let X be a regular code (resp., prefix set). Then $\mathcal{A}_X$ is an unambiguous (resp., deterministic) automaton. Conversely, let $\mathcal{A}$ be an automaton such that $L_{\mathcal{A}} = X^*$ where $X \cap X^2X^* = \emptyset$. If $\mathcal{A}$ is unambiguous (resp., deterministic), then X is a code (resp. a prefix set).*

Incompletable words of a regular set are characterized by the following.

Lemma 3. *Let X be a regular set and $\mathcal{A} = \langle Q, A, \delta, 1\rangle$ be a transitive automaton accepting X^*. Then $w \in A^*$ is a completable word of X if and only if $Qw \neq \emptyset$.*

In particular, X is complete if and only if $\mathcal{A}$ is complete.

Similarly it is possible to characterize synchronizing pairs of regular codes [6].

Lemma 4. *Let X be a regular code and $\mathcal{A} = \langle Q, A, \delta, 1\rangle$ be a transitive unambiguous automaton accepting X^* and $w_1, w_2 \in A^*$. Then (w_1, w_2) is a synchronizing pair of X if and only if $w_1w_2 \in X^*$ and $Qw_1 \cap Qw_2{}^{-1} = \{1\}$.*

Consequently, X is synchronizing if and only if $\mathcal{A}$ is synchronizing.

3 The main result

The main result of this paper is related to a problem that was formulated in [18] by Restivo. Let $\mathcal{L}$ be a class of finite languages. For all $n > 0$ we set

$$R_{\mathcal{L}}(n) = \sup_{d \geq 1} R_{\mathcal{L}}(n, d)\,, \quad C_{\mathcal{L}}(n) = \sup_{d \geq 1} C_{\mathcal{L}}(n, d)\,.$$

In [18], it was conjectured that if $\mathcal{F}$ is the class of all finite languages, then $R_{\mathcal{F}}(n) \leq 2n^2$. If we restrict ourselves to prefix codes, we get

Proposition 2. *([18]) Let $\mathcal{P}$ be the class of finite prefix codes. Then*

$$R_{\mathcal{P}}(n) \leq 2n^2.$$

However, in the general case, the previous bound was disproved in [14]. A more general and larger counterexample is given in [15]. We can thus state a slightly weaker version of the problem as follows.

Conjecture 1. (Restivo's Conjecture) Let $\mathcal{F}$ be the class of all finite languages. Then $R_{\mathcal{F}}(n) = O(n^2)$.

In this context, the main result of this paper is the following.

Proposition 3. *Let $\mathcal{M}$ be the class of complete finite codes. For all $n, d > 0$,*

$$C_{\mathcal{M}}(n, d) \leq 2R_{\mathcal{F}}(n, d + 1) + 2n - 2.$$

Before proving Proposition 3, it is convenient to discuss some interesting consequences of this result. First, if Restivo's conjecture is true, we get

$$C_{\mathcal{M}}(n, d) = O(n^2).$$

Moreover, the bound above would be sharp, as we explain below. Consider the prefix code $X_n = aA^{n-1} \cup bA^{n-2}$ on the alphabet $A = \{a, b\}$. The minimal automaton accepting X_n^* has been studied in [1], where it has been proved that the minimal length of its synchronizing words is $n^2 - 3n + 3$. From this, one derives that any synchronizing pair (w_1, w_2) of X_n verifies $|w_1 w_2| \geq (n-1)^2$. In particular, a synchronizing pair of X_n of minimal length is $((ab^{n-2})^{n-1}, \epsilon)$. This provides the lower bound

$$\mathcal{C}_{\mathcal{M}}(n, 2) \geq \mathcal{C}_{\mathcal{P}}(n, 2) \geq (n-1)^2,$$

for the parameter $\mathcal{C}_{\mathcal{M}}(n, 2)$.

It is also worth to do a remark on a recent result by Béal and Perrin. In [2] (cf. also [3]), it is proved that a synchronizing complete prefix code X with n code-words has a synchronizing word of length $2(n-2)(n-3)+1$. This result is derived from an upper bound to the length of shortest synchronizing words of synchronizing one-cluster automata. However, in view of Proposition 3 and Restivo's conjecture, this bound seems of no help in obtaining a good evaluation of the parameter $C_{\mathcal{P}}(n, 2)$, as one may have $n \simeq 2^{\ell(X)}$. This suggests that a bound in term of the size of X may be more informative than a bound in terms of the cardinality.

3.1 Proof of Proposition 3

Let us now proceed to prove Proposition 3. For this purpose, let X be a finite complete synchronizing code over a d-letter alphabet A and let $n = \ell(X)$. Let $\mathcal{A}_X = \langle Q, A, \delta, 1\rangle$ be the unambiguous automaton that accepts X^*. The proof of Proposition 3 is based upon the following lemma.

Lemma 5. *Let (v_1, v_2) be a synchronizing pair of X. Then, there exist words $w_1, w_2 \in A^*$ such that*

$$|w_1|, |w_2| \leq R_{\mathcal{F}}(n, d+1), \quad Qw_1 \subseteq Qv_1, \quad Q{w_2}^{-1} \subseteq Qv_2^{-1}.$$

Indeed, assume that Lemma 5 holds. As X is complete, the word w_1w_2 has an X-completion (r, s). With no loss of generality, we may suppose that $|r|, |s| \leq n-1$. Since (v_1, v_2) is a synchronizing pair, one has

$$Q(rw_1) \cap Q(w_2s)^{-1} \subseteq Qw_1 \cap Q{w_2}^{-1} \subseteq Qv_1 \cap Q{v_2}^{-1} = \{1\}.$$

Moreover, the word $rw_1w_2s \in X^*$ is accepted by $\mathcal{A}_X$ and therefore there is a state $q \in Q$ such that $q \in 1rw_1$ and $1 \in qw_2s$. Thus, $q \in Q(rw_1) \cap Q(w_2s)^{-1} \subseteq \{1\}$, that is, $q = 1$. This proves that $rw_1, w_2s \in X^*$ and by Lemma 4 (rw_1, w_2s) is a synchronizing pair of X.

Now, our main goal is to prove Lemma 5. For the sake of simplicity, we will prove the existence of the word w_1 that fulfills the conditions of Lemma 5 since the proof of the existence of the word w_2 can be obtained by using a symmetric construction. The main tool of this proof is a new automaton we construct below.

Let (v_1, v_2) be a synchronizing pair of X. If $v_1 = \epsilon$, the statement is trivially verified by $w_1 = v_1$. Thus we assume $v_1 \neq \epsilon$ and set $v_1 = ua$, with $u \in A^*$ and $a \in A$.

Let a' be a symbol not belonging to A and let $A' = A \cup \{a'\}$. We consider a new automaton $\mathcal{A}' = \langle Q, A', \delta', 1\rangle$ where the transition map δ' is defined as follows: for every $q \in Q$ and $a \in A$, $\delta'(q, a) = \delta(q, a)$ and

$$\delta'(q, a') = \begin{cases} \delta(q,a) \cup \{1\} & \text{if } q \notin \delta(Q, u), \\ \delta(q,a) \setminus \{1\} & \text{if } q \in \delta(Q, u). \end{cases} \tag{1}$$

It is useful to remark that, by construction, the automaton $\mathcal{A}'$ is still transitive. Let Y be the minimal generating set of the language accepted by $\mathcal{A}'$. Thus, $L_{\mathcal{A}'} = Y^*$ and $Y \cap Y^2Y^* = \emptyset$.

Lemma 6. *The set Y is incomplete.*

Proof. By (1) one has $\delta'(Q, ua') = \delta(Q, ua)\setminus\{1\} = \delta(Q, v_1)\setminus\{1\}$ and $\delta'(Q, v_2^{-1}) = \delta(Q, v_2)$. Taking into account that (v_1, v_2) is a synchronizing pair of X, one derives

$$\delta'(Q, ua') \cap \delta'(Q, v_2^{-1}) = (\delta(Q, v_1) \cap \delta'(Q, v_2^{-1})) \setminus \{1\} = \emptyset\,.$$

It follows that $\delta'(Q, ua'v_2) = \emptyset$. This equation proves that the automaton $\mathcal{A}$ is not complete. Thus, by Lemma 3, Y is an incomplete set. $\square$

Lemma 7. *It holds that $\ell(Y) \leq \ell(X)$.*

Proof. In order to prove the statement, it is enough to show that, for every $y \in Y$, there exists $x \in X$ with $|y| \leq |x|$.

Let $y = a_1 \cdots a_k \in Y$, with $a_i \in A'$, for $i = 1, \ldots, k$. Since $Y \cap Y^2Y^* = \emptyset$, in the graph of $\mathcal{A}'$ there is a path

$$c' = 1 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_k \xrightarrow{a_k} 1,$$

where, for every $i = 1, \ldots, k$, $q_i \neq 1$. Let us now construct a path c in the graph of $\mathcal{A}_X$ such that $||c|| = x \in X$, with $|x| \geq |y|$, so completing the proof.

By the definition of $\mathcal{A}'$, any edge $p \xrightarrow{b} q$ of the graph of $\mathcal{A}'$ with $b \neq a'$ is also an edge of the graph of $\mathcal{A}$. Moreover, if $p \xrightarrow{a'} q$ is an edge of the graph of $\mathcal{A}'$ with $q \neq 1$, then $p \xrightarrow{a} q$ is an edge of the graph of $\mathcal{A}$. Thus, by replacing in c', every transition $q_i \xrightarrow{a'} q_{i+1}$, by $q_i \xrightarrow{a} q_{i+1}$ and deleting the last edge $q_k \xrightarrow{a_k} 1$, we find a path

$$d = 1 \xrightarrow{b_1} q_1 \xrightarrow{b_2} q_2 \cdots \xrightarrow{b_{k-1}} q_k$$

of the graph of $\mathcal{A}$. Since $\mathcal{A}$ is transitive, one can catenate d with a simple path from q_k to 1. In such a way, we obtain a path c of the graph of $\mathcal{A}$ starting and ending in 1, with all intermediate states distinct from 1 and length $\geq k + 1$. As is well known, as $\mathcal{A}$ is unambiguous, the label x of such a path is a word of the minimal generating set X of X^*. Since $|x| \geq k + 1 = |y|$, this completes the proof. □

We now prove the following lemma.

Lemma 8. *Let v be an incompletable word of Y of minimal length. There exists a word $w_1 \in A^*$ such that*

$$|w_1| \leq |v|, \quad Qw_1 \subseteq Qv_1.$$

Proof. Let v be an incompletable word of Y of minimal length, with the number $|v|_{a'}$ as small as possible. Then, by Lemma 3, one has $\delta'(Q, v) = \emptyset$.

The letter a' necessarily occurs in v, since by the completeness of $\mathcal{A}$, one has $\delta'(Q, r) = \delta(Q, r) \neq \emptyset$ for all $r \in A^*$. Thus, we can write $v = u_1a'u_2$, with $u_1 \in A^*$ and $u_2 \in A'^*$.

Let us verify that $\delta(Q, u_1) \subseteq \delta(Q, u)$. Indeed, suppose the contrary. Then, by (1), one has

$$\delta'(Q, u_1a') = \delta(Q, u_1a) \cup \{1\} = \delta'(Q, u_1a) \cup \{1\}$$

and consequently, $\delta'(Q, u_1au_2) \subseteq \delta'(Q, u_1a'u_2) = \emptyset$. Thus, u_1au_2 is an incompletable word of Y, but this contradicts the minimality of $|v|_{a'}$.

We conclude that $\delta(Q, u_1) \subseteq \delta(Q, u)$ and therefore taking $w_1 = u_1a$ and recalling that $v_1 = ua$, one has $\delta(Q, w_1) \subseteq \delta(Q, v_1)$ and $|w_1| \leq |v|$. The statement follows. □

Let us finally remark that Lemma 7 and Lemma 8 yield

$$|w_1| \leq R_{\mathcal{F}}(n, d+1), \quad Qw_1 \subseteq Qv_1.$$

The proof of Lemma 5 is thus complete.

4 Reduction to the binary case

The aim of this section is to study how much the parameters $R_{\mathcal{L}}(n,d)$ and $C_{\mathcal{L}}(n,d)$ vary according to the cardinal number d of the alphabet. We start to analyze the parameter $R_{\mathcal{L}}(n,d)$. In the sequel, B denotes the binary alphabet $B = \{a, b\}$. The following lemmas are needed for the proof of Proposition 4.

Lemma 9. *Let $Y \subseteq A^*$ be a complete finite set. Then any word w of A^* has a Y-completion (y, s) with $y \in Y^*$.*

Lemma 10. *Let $h : A^* \to B^*$ be a prefix encoding and $Y \subseteq A^*$. The set $h(Y)$ is complete if and only if Y and $h(A)$ are complete.*

Encoding a d-letter alphabet on a suitable complete binary prefix code one obtains

Proposition 4. *Let $\mathcal{L}$ be the class of finite languages (resp. codes). Then*

$$R_{\mathcal{L}}(n,d) \leq \left\lceil \frac{R_{\mathcal{L}}(\lceil \log_2 d \rceil n, 2)}{\lfloor \log_2 d \rfloor} \right\rceil . \tag{2}$$

Proof. Let A be a d-letter alphabet and let X be a finite incompletable language over A of size n. By Theorems 2 and 3, there exists a maximal prefix code Y over B such that $\mathrm{Card}(Y) = d$ and, for every $y \in Y$, $\lfloor \log_2 d \rfloor \leq |y| \leq \lceil \log_2 d \rceil$. Let $h : A^* \to B^*$ be a monomorphism constructed by Y. In particular, for every $a \in A$, we have

$$\lfloor \log_2 d \rfloor \leq |h(a)| \leq \lceil \log_2 d \rceil . \tag{3}$$

By (3) the size of $h(X)$ is not greater than $n\lceil \log_2 d \rceil$. By Lemma 10, since X is incompletable, $h(X)$ is incompletable as well. Let v be an incompletable word in $h(X)$ of minimal length. Hence we have

$$|v| \leq R_{\mathcal{L}}(\lceil \log_2 d \rceil n, 2). \tag{4}$$

Since $h(A)$ is a complete prefix code, by Theorem 4, there exist $u \in A^*$ and $s \in B^*$ such that $h(u) = vs$ and by (3)

$$|u| \leq \left\lceil \frac{|v|}{\lfloor \log_2 d \rfloor} \right\rceil . \tag{5}$$

Let us check that u is incompletable. By contradiction, deny. Then $r'us' \in X^*$, for some $r', s' \in A^*$. Consequently, $h(r'us') = h(r')vsh(s') \in h(X^*)$, thus implying that v is completable in $h(X)$. Now (2) easily follows from the latter, (4) and (5). □

Let us now analyze the parameter $C_{\mathcal{L}}(n,d)$. The next two lemmas are useful for this purpose. In particular the following lemma is algebraically similar to Lemma 10.

Lemma 11. *Let $h : A^* \to B^*$ be a monomorphism and let $Y \subseteq A^*$ be a complete set. The set $h(Y)$ is synchronizing if and only if Y and $h(A)$ are synchronizing.*

Lemma 12. *Let $k_1, \dots, k_n, d > 0$ be such that*

$$\gcd(k_1, k_2, \dots, k_n) = 1\,, \qquad \sum_{i=1}^{n} d^{-k_i} = 1.$$

Then $k_1, \dots, k_n$ are the code-word lengths of a synchronizing complete prefix code over d letters.

Remark 1. It is worth noticing that a finite synchronizing complete prefix code over d letters satisfies both the conditions of Lemma 12. Indeed, by Theorem 3, if $X = \{x_1, \dots, x_n\}$ is a complete code over d letters, one gets $\sum_{i=1}^{n} d^{-k_i} = 1$. Moreover, by Theorem 5, there exists a constant $x \in X^*$ for X such that $xA^*x \subseteq X^*$. Let $\gamma = \gcd(k_1, k_2, \dots, k_n)$. Since, by the latter result, $x^2, xax \in X^*$, with $a \in A$, γ should divide both $2|x|$ and $2|x|+1$, whence $\gamma = 1$.

As an application of the two lemmas above, encoding a d-letter alphabet on a suitable complete binary synchronizing code, one obtains the following result:

Proposition 5. *Let $\mathcal{L}$ be the class of finite complete languages (resp. codes, prefix codes). Then*

$$C_{\mathcal{L}}(n,d) \leq \left\lceil \frac{C_{\mathcal{L}}(\lceil \log_2(d+1) \rceil n, 2)}{\lfloor \log_2(d-1) \rfloor} \right\rceil . \tag{6}$$

A similar bound can be found also in the case where completeness is not required:

Proposition 6. *Let $\mathcal{L}$ be the class of finite languages (resp. codes, prefix codes). Then*

$$C_{\mathcal{L}}(n,d) \leq \left\lceil \frac{C_{\mathcal{L}}(\lceil \log_2(d+1) \rceil n, 2)}{\lceil \log_2(d+1) \rceil} \right\rceil . \tag{7}$$

References

1. D. S. Ananichev, V. V. Gusev, M. V. Volkov, Slowly synchronizing automata and digraphs, in: P. Hliněný, A. Kučera eds., *MFCS 2010 Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci.* Vol. 6281, pp. 55–65, Springer, Berlin, 2010.
2. M.-P. Béal, D. Perrin, A quadratic upper bound on the size of a synchronizing word in one-cluster automata, in: V. Diekert, D. Nowotka eds., *DLT 2009 Developments in Language Theory, Lecture Notes in Computer Science*, Vol. 5583, pp. 81–90, Springer, Berlin, 2009.

3. M.-P. Béal, M. V. Berlinkov, D. Perrin, A quadratic upper bound on the size of a synchronizing word in one-cluster automata, *Int. J. Found. Comput. Sci.,* 22, 277–288, 2011.
4. J. Berstel, D. Perrin, Ch. Reutenauer, Codes and Automata, Encyclopedia of Mathematics and its Applications, 129, Cambridge University Press, 2009.
5. J. M. Boë, A. de Luca, A. Restivo, Minimal complete sets of words, *Theoret. Comput. Sci.,* 12, 325–332, 1980.
6. A. Carpi, On synchronizing unambiguous automata, *Theoret. Comput. Sci.,* 60, 285–296, 1988.
7. A. Carpi, F. D'Alessandro, The Synchronization Problem for Strongly Transitive Automata, in: M. Ito, M. Toyama eds., *DLT 2008 Developments in Language Theory, Lecture Notes in Computer Science*, Vol. 5257, pp. 240–251, Springer, Berlin, 2008.
8. A. Carpi, F. D'Alessandro, Strongly transitive automata and the Černý conjecture *Acta Informatica,* 46, 591–607, 2009.
9. A. Carpi, F. D'Alessandro, The synchronization problem for locally strongly transitive automata, in: R. Královič, D. Niwiński eds., *MFCS 2009 Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci.,* Vol. 5734, pp. 211–222, Springer, Berlin, 2009.
10. A. Carpi, F. D'Alessandro, On the Hybrid Černý-Road coloring problem and Hamiltonian paths, in: Y. Gao, H. Lu, S. Seki, S. Yu eds., *DLT 2010 Developments in Language Theory, Lecture Notes in Comput. Sci.,* Vol. 6224, pp. 124–135, Springer, Berlin, 2010.
11. A. Carpi, F. D'Alessandro, Independent sets of words and the synchronization problem, *Advances in Applied Mathematics,* 50, 339–355, 2013.
12. J. Černý, Poznámka k. homogénnym experimenton s konečnými automatmi, *Mat. fyz. cas SAV,* 14, 208–215, 1964.
13. A. de Luca, F. D'Alessandro, Teoria degli Automi Finiti, 68, Springer Italia, 2013.
14. G. Fici, E. V. Pribavkina, J. Sakarovitch, On the Minimal Uncompletable Word Problem, CoRR, arXiv: 1002.1928, 2010.
15. V. V. Gusev, E. V. Pribavkina, On Non-complete Sets and Restivo's Conjecture, in: G. Mauri, A. Leporati eds., *DLT 2011 Developments in Language Theory, Lecture Notes in Comput. Sci.,* Vol. 6795, pp. 239–250, Springer, Berlin, 2011.
16. J. E. Pin, *Le problème de la synchronization et la conjecture de Cerny,* Thèse de 3ème cycle, Université de Paris 6, 1978.
17. J. E. Pin, Sur un cas particulier de la conjecture de Cerny, in: G. Ausiello, C. Böhm eds., *5th ICALP Lecture Notes in Computer Science*, Vol. 62, pp. 345–352, Springer, Berlin, 1978.
18. A. Restivo, Some remarks on complete subsets of a free monoid, in: A. de Luca ed., *Non-Commutative Structures in Algebra and Geometric Combinatorics, International Colloquium, Arco Felice, July 1978*, *Quaderni de "La Ricerca Scientifica"*, CNR, 109, 19–25, 1981.
19. A. N. Trahtman, The road coloring problem, *Israel J. Math.,* 172, 51–60, 2009.
20. M. V. Volkov, Synchronizing automata and the Cerny conjecture, in: C. Martín-Vide, F. Otto, H. Fernau eds., *LATA 2008 Language and Automata Theory and Applications, Lecture Notes in Comput. Sci.,* Vol. 5196, pp. 11–27, Springer, Berlin, 2008.

Reasoning about connectivity without paths[⋆]

Alberto Casagrande
acasagrande@units.it

Eugenio Omodeo
eomodeo@units.it

Dep. of Mathematics and Geosciences
University of Trieste
Trieste, Italy

Abstract. In graph theory connectivity is stated, prevailingly, in terms of paths. While exploiting a proof assistant to check formal reasoning about graphs, we chose to work with an alternative characterization of connectivity: for, within the framework of the underlying set theory, it requires virtually no preparatory notions.
We say that a graphs devoid of isolated vertices is *connected* if no subset of its set of edges, other than the empty set and the set of all edges, is vertex disjoint from its complementary set. Before we can work with this notion smoothly, we must prove that every connected graph has a *non-cut* vertex, i.e., a vertex whose removal does not disrupt connectivity.
This paper presents such a proof in accurate formal terms and copes with *hypergraphs* to achieve greater generality.

1 Introduction

Connectivity plays a crucial role not only in graph theory, but also in topology. The number of connected components of a graph is a topological invariant, it corresponds to the multiplicity of the eigenvalue 0 in the Laplacian matrix that represents the graph, and, in the recent years, it has been related to the number of claw-free subgraphs of the graph itself [1]. Because of the ubiquity of this notion, it deserves an autonomous and insightful treatment in a large scale formalization effort, such as the one envisioned in [8] or in [6].

Non-cut vertices are vertices whose removal preserves the graph connectivity. The notion of connectivity is traditionally given in term of *paths*, and in such terms one proves that any graph contains non-cut vertices. These vertices are largely used in inductive proofs over connected graphs: the pattern is to apply the inductive hypothesis to a graph deprived of a non-cut vertex and, then, to prove that the investigated property is retained when the vertex is reinstated.

While formally defining the notion of path is not really a problem, from a foundational point of view, it appears to be an *out-of-focus* effort in this context: it would in fact bring into play notions (e.g., natural numbers) which are barely related to the theme of discourse.

⋆ This work has been partially supported by Istituto Nazionale di Alta Matematica (INdAM).

This paper exploits a path-free notion of connectivity to formally prove that *connected undirected hypergraphs* are always endowed with non-cut vertices.

Our immediate motivation for undertaking this study on connectivity is a formalization task that has been successfully carried out recently with the assistance of the proof checker Referee/Ætnanova [6]: as reported in [5], taking advantage of the Milanič and Tomescu representation theorem for *connected claw-free graphs* [4], we could achieve with relative ease the proof that any such graph owns a near-perfect matching and has a Hamiltonian cycle in its square. We took it for granted that every connected graph has a spanning tree. This left us with a proof obligation, and since the existence of a spanning tree plainly reduces to the proof that every connected graph has a non-cut vertex, we are now beginning to fill the gap, with the formalization task on which we will report below.

This paper is organized as follows: Section 2 introduces the notation and all the needed notions. Section 3 formalizes the result aimed at and splits the proof of it into multiple steps that are detailed in Sections 3.1, 3.2, and 3.3 (some basic properties and technical lemmas are proved in the appendix). Finally, Section 4 draws our conclusions and suggests future work.

2 Preliminaries

We work with in the Zermelo-Fraenkel set theory (ZF) and all the notions treated in this paper are defined through it. Besides the standard Boolean propositional functions (i.e., $\wedge$, $\neg$), the used formal language provides intersection ($\cap$), union ($\cup$), and difference ($\setminus$) over sets as well as both the membership ($\in$) and inclusion ($\supseteq$) relations. From a formal point of view, the Boolean functions $\vee$ and $\rightarrow$ and the relations over sets $\notin$, $\supsetneq$, $\subseteq$, $\subsetneq$, $=$, and $\neq$ are shortcuts for non-atomic formulæ whose semantics is the standard one. While the notion of *cardinality of a set* is not formally included in the adopted language, it is worthwhile to introduce the relation $|\cdot|_{\geq n}$ that associates each finite set with the number of elements belonging to it. This relation is not really necessary, but it enables more natural definitions for the subsequent notions. The relation $|\cdot|_{\geq n}$ is defined as:

$$|S|_{\geq n} :- \begin{cases} \top & \text{if } n = 0 \\ \exists v \in S \ |S \setminus \{v\}|_{\geq n-1} & \text{otherwise} \end{cases}$$

It is easy to see that, for any natural number $n \in \mathbb{N}$ and any set S, $|S|_{\geq n}$ holds if and only if $|S| \geq n$.

We characterize finitude as proposed by Tarski [7]: a set S is finite if and only if every not empty class of subsets of S contains an element which is minimal with respect to $\subseteq$. This clue is captured by the following definition

$$\textbf{Finite}\,(S) :- \forall P \in \wp(\wp(S)) \setminus \{\emptyset\} \ \exists M \ \wp(M) \cap P = \{M\}.$$

Tarski himself proved that if S is finite then every not empty class of subsets of S contains also a maximal element [7].

The basic notions of *(hyper)graph*, *edge*, and *node* are defined as follows.

Definiton 1 *An* edge *is a finite set endowed with at least two elements. A* hypergraph *G is a finite set of edges, i.e.,* $\boldsymbol{Graph}(G) :- \boldsymbol{Finite}(G) \wedge \forall e \in G\,(|e|_{\geq 2} \wedge \boldsymbol{Finite}(e))$. *The elements of the edges of G are called* nodes, *or* vertices, *of G.*

By standard terminology, the word graph refers to hypergraphs whose edges have cardinality 2. However, we take the freedom to abbreviate "hypergraph" into "graph" because this work deals exclusively with the more general notion.

In accordance with the literature (e.g., see [2, 3]), our definition does not allow graphs to have self-loops, namely singleton edges, and it explicitly requires that each of the edges contains at least 2 distinct elements.

Any subset of a graph is a graph.

Lemma 1 $P \subseteq G \wedge \boldsymbol{Graph}(G) \rightarrow \boldsymbol{Graph}(P)$

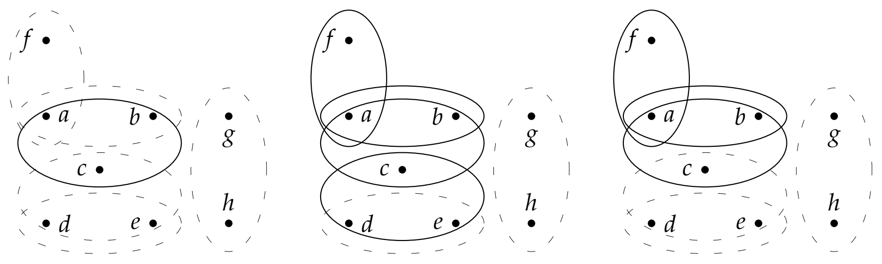
Proof. The claim follows directly from the definition of $\mathbf{Graph}(\cdot)$.

Let $\boldsymbol{Nodes}(G)$, $\boldsymbol{Cov}(G, P)$, and $\boldsymbol{Contains}(G, v)$ represent the set of nodes of G, the set of edges in G that share nodes with any edge in P, and the set of edges in G that contain the node v. More formally,

$$\boldsymbol{Nodes}(G) \stackrel{\text{def}}{=} \bigcup_{e \in G} e \qquad\qquad \boldsymbol{Cov}(G, P) \stackrel{\text{def}}{=} \{e \in G \,|\, e \cap \boldsymbol{Nodes}(P) \neq \emptyset\}$$

$$\boldsymbol{Contains}(G, v) \stackrel{\text{def}}{=} \{e \in G \mid v \in e\}$$

If $\boldsymbol{Contains}(G, v)$ is a singleton, then v is said to be a *boundary vertex*.



(a) A graph $P \subseteq G$ (b) $\boldsymbol{Cov}(G, P)$ (c) $\boldsymbol{Contains}(G, a)$

Fig. 1: In above figures, ovals represent the elements of G.

If, for every nonnull set $P \subsetneq G$, P shares some nodes with the graph $G \setminus P$, then G is said to be *connected* (in short, $\mathbf{Conn}(G)$).

$$\mathbf{Conn}(G) :- \mathbf{Graph}(G) \wedge \forall P\ (\emptyset \subsetneq P \subsetneq G \rightarrow \boldsymbol{Nodes}(P) \cap \boldsymbol{Nodes}(G \setminus P) \neq \emptyset)$$

If the graph G is connected and $G \setminus \{e\}$ is not connected, then e is said to be a *cutting edge*.

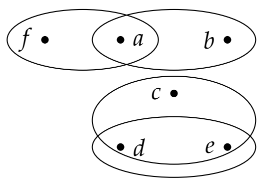


Fig. 2: Let G and P be $\{\{a,b\},\{a,f\},\{c,d,e\},\{d,e\}\}$ and $\{\{a,b\},\{a,f\}\}$, respectively. The graph G is not connected as $\textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(G \setminus P)$ is empty and $P \subsetneq G$.

Let us notice that it is not always the case that by removing a node v from all edges of a graph G we get a graph. As a matter of fact, some of the edges of G that contain v may have cardinality 2. If we remove v from such edges, we obtain sets whose cardinality is 1 and, by definition of edge, these are not edges. For instance, if G contains an edge $\{a,b\}$ and v is a, then $\{a,b\} \setminus \{a\}$ is $\{b\}$ which is not an edge because has cardinality 1.

Filter (G,v) is the set obtained by first removing v from all edges of G and then filtering out all the resulting sets whose cardinality is less than 2. Since all the elements of ***Filter*** (G,v) have cardinality 2 at least, ***Filter*** (G,v) is a graph by definition of graph.

$$\textbf{\textit{Filter}}(G,v) \overset{\text{def}}{=} \{e \setminus \{v\} \mid e \in G \wedge |e \setminus \{v\}|_{\geq 2}\}$$

Notice that, if $\{v,w\}$ is the only edge in G that contains w, then w does not belong to ***Nodes*** (***Filter*** (G,v)). When we write ***Nodes*** (G), ***Cov*** (G,P), ***Contains*** (G,v), or ***Filter*** (G,v), we implicitly assume that both of G and P are graphs.

Let G and v be a graph and a node, respectively. We define ***Lost*** (G,v) to be the set of nodes of G that are not nodes of ***Filter*** (G,v) and differ from v.

$$\textbf{\textit{Lost}}(G,v) \overset{\text{def}}{=} \textbf{\textit{Nodes}}(G) \setminus (\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) \cup \{v\})$$

If ***Lost*** (G,v) is nonnull, then we say that v is a *losing vertex* of G and all of its elements are said to be *lost by v*.

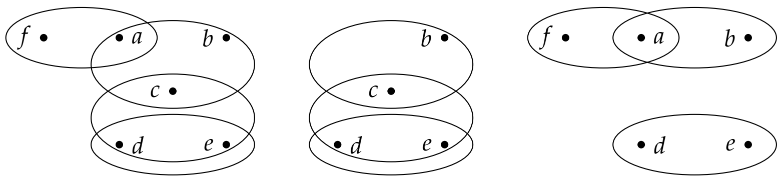
Whenever G is connected and either ***Filter*** (G,v) is not connected or some of the nodes in G other than v do not belong to ***Filter*** (G,v), v is a *cut vertex* of G (**Cutting** (G,v) holds).

$$\textbf{Cutting}(G,v) :\!-\textbf{Conn}(G) \wedge |G|_{\geq 2} \wedge (\neg\textbf{Conn}(\textbf{\textit{Filter}}(G,v)) \vee \textbf{\textit{Lost}}(G,v) \neq \emptyset)$$

We call *non-cut vertex* any vertex that is not a cut vertex.

3 Hypergraphs have non-cut vertices

Our goal is to provide a proof that every connected hypergraph contains a non-cut vertex. This is encoded by the following corollary.



(a) A graph G (b) $\textbf{\textit{Filter}}(G,a)$ (c) $\textbf{\textit{Filter}}(G,c)$

Fig. 3: The property $\textbf{Cutting}(G,v)$ holds whenever $\textbf{\textit{Filter}}(G,v)$ is not connected or it contains fewer nodes than $\textbf{\textit{Nodes}}(G) \setminus \{v\}$. Both $\textbf{Cutting}(G,a)$ and $\textbf{Cutting}(G,c)$ do hold since $\textbf{\textit{Filter}}(G,a)$ lost the node f (see Fig. 3b) and $\textbf{\textit{Filter}}(G,c)$ is not connected (see Fig. 3c).

Corollary 1 $\textbf{\textit{Conn}}(G) \rightarrow (G = \emptyset \vee \exists v \in \textbf{\textit{Nodes}}(G)\ \neg\textbf{\textit{Cutting}}(G,v))$

We split the proof of above corollary into the proofs of two statements: (1) if G has a losing vertex, then G has is a non-cut vertex (see Section 3.1); (2) any connected hypergraph G contains a node v such that $\textbf{\textit{Filter}}(G,v)$ is connected (see Section 3.3). By definition of $\textbf{Cutting}(\cdot)$, this suffices to yield the claim of Corollary 1.

3.1 Losing vertices yield non-cut vertices

In this section, we prove that, if G has a losing vertex, then it has a non-cut vertex too. The following theorem formalizes this statement.

Theorem 1 $(\textbf{\textit{Graph}}(G) \wedge \textbf{\textit{Lost}}(G,v) \neq \emptyset) \rightarrow \exists v' \in \textbf{\textit{Nodes}}(G)\ \neg\textbf{\textit{Cutting}}(G,v')$

First of all we need to prove that v is a losing vertex if and only if there exists a vertex lost by v such that $\textbf{\textit{Contains}}(G,v') = \{\{v,v'\}\}$. By definition of $\textbf{\textit{Filter}}(\cdot)$, $\textbf{\textit{Filter}}(G,v)$ contains all the set $e \setminus \{v\}$ such that e is an edge of G and $e \setminus \{v\}$ has at least cardinality 2. Hence, $v' \neq v$ is a node of G and it is not a node of $\textbf{\textit{Filter}}(G,v)$ if and only if there exists an edge $e \in G$ such that $|e \setminus \{v\}| < 2$. Since e has at least cardinality 2 by definition of $\textbf{Graph}(\cdot)$, we can conclude that e equals $\{v,v'\}$.

Lemma 2 $\textbf{\textit{Graph}}(G) \rightarrow (v' \in \textbf{\textit{Lost}}(G,v) \leftrightarrow \textbf{\textit{Contains}}(G,v') = \{\{v,v'\}\})$

Proof. ($\rightarrow$) By definition of $\textbf{\textit{Nodes}}(\cdot)$, if $v' \in \textbf{\textit{Nodes}}(G)$, then there exists $e \in G$ such that $v' \in e$. Analogously, if $v' \notin \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v))$, then $v' \notin e'$ for all $e' \in \textbf{\textit{Filter}}(G,v)$. However, $\textbf{\textit{Filter}}(G,v) = \{e \setminus \{v\} \mid e \in G \wedge |e \setminus \{v\}|_{\geq 2}\}$ by definition. Hence, if $v' \notin \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v))$, either $v' \notin e \setminus \{v\}$ or $|e \setminus \{v\}| < 2$ for all $e \in G$ and, if $v' \notin \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v))$ and $v' \in \textbf{\textit{Nodes}}(G)$, then either $v' = v$ or $|e \setminus \{v\}| < 2$ for all $e \in G$ such that $v' \in e$. Since $|e|_{\geq 2}$ by definition of $\textbf{Graph}(\cdot)$ and $v' \in \textbf{\textit{Nodes}}(G) \setminus (\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) \cup \{v\})$ holds by hypothesis, $e = \{v,v'\}$ for all $e \in G$ such that $v' \in e$. Moreover, $\{v,v'\} \in G$ because if $v' \in \textbf{\textit{Nodes}}(G) \setminus (\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) \cup \{v\})$, then $v' \in \textbf{\textit{Nodes}}(G)$ and there exists $e \in G$ such that $v' \in e$. By definition of $\textbf{\textit{Contains}}(\cdot)$, it follows that $\textbf{\textit{Contains}}(G,v') = \{\{v,v'\}\}$.

($\leftarrow$) Let us assume that $\boldsymbol{Contains}(G, v') = \{\{v, v'\}\}$. By definition of $\mathbf{Graph}(\cdot)$, if $e \in G$, then $|e|_{\geq 2}$. By definition of $\boldsymbol{Contains}(\cdot)$, $\boldsymbol{Contains}(G, v') \subseteq G$ and, thus, $\{v, v'\} \in G$ and $v \neq v'$. Moreover, $v' \in \boldsymbol{Nodes}(G)$ by definition of $\boldsymbol{Nodes}(\cdot)$ and $v' \notin e$ for all $e \in G \setminus \{\{v, v'\}\}$ by definition of $\boldsymbol{Contains}(\cdot)$. By definition of $\boldsymbol{Filter}(\cdot)$, $\boldsymbol{Filter}(G, v)$ is the set $\{e \setminus \{v\} \mid e \in G \wedge |e \setminus \{v\}|_{\geq 2}\}$. Since $\boldsymbol{Contains}(G, v') = \{\{v, v'\}\}$ by hypothesis, if $|e \setminus \{v\}|_{\geq 2}$ holds, then $v' \notin e$. Thus, v' belongs neither to $\boldsymbol{Filter}(G, v)$ nor to $\boldsymbol{Filter}(G, v) \cup \{v\}$ because $v \neq v'$ and $v' \in \boldsymbol{Nodes}(G) \setminus (\boldsymbol{Filter}(G, v) \cup \{v\})$. □

Our next step is to prove that any boundary vertex is non-cut. Since v belongs to a single edge, for every subset P of G, either $v \in \boldsymbol{Nodes}(P)$ and $v \notin \boldsymbol{Nodes}(G \setminus P)$ or $v \notin \boldsymbol{Nodes}(P)$ and $v \in \boldsymbol{Nodes}(G \setminus P)$. In either case, if the set $\boldsymbol{Nodes}(P) \cap \boldsymbol{Nodes}(G \setminus P)$ is nonnull, so is $(\boldsymbol{Nodes}(P) \cap \boldsymbol{Nodes}(G \setminus P)) \setminus \{v\}$. If G is connected, either $G \setminus e$ is empty or v' belongs to another edge of G. In the former case, $G \setminus e$ is connected. In the latter, $\boldsymbol{Nodes}(G) \setminus \{v\} = \boldsymbol{Nodes}(G \setminus e)$ and $G \setminus e$ must be connected. Thus, if G is connected and v is contained exclusively by an edge e whose cardinality is 2, then $G \setminus e$ is still connected.

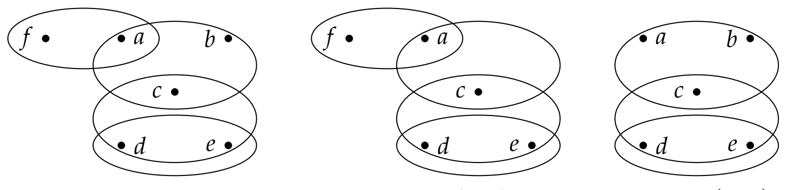
Lemma 3 $(\boldsymbol{Conn}(G) \wedge \boldsymbol{Contains}(G, v) = \{\{v, v'\}\}) \rightarrow \boldsymbol{Conn}(G \setminus \boldsymbol{Contains}(G, v))$

Proof. Let us assume that there exist G_0, v_0, and v_0' such that both the formulæ $\mathbf{Conn}(G_0)$ and $\boldsymbol{Contains}(G_0, v_0) = \{\{v_0, v_0'\}\}$ hold, while the formula $\mathbf{Conn}(G_0 \setminus \boldsymbol{Contains}(G_0, v_0))$ does not. By definition of $\mathbf{Conn}(\cdot)$, there exists a nonnull set P_0 such that P_0 is a proper subset of $G_0 \setminus \boldsymbol{Contains}(G_0, v_0)$ and $\boldsymbol{Nodes}(P_0) \cap \boldsymbol{Nodes}((G_0 \setminus \boldsymbol{Contains}(G_0, v_0)) \setminus P_0) = \emptyset$. Thus, the set $\boldsymbol{Nodes}(P_0) \cap \boldsymbol{Nodes}((G_0 \setminus P_0) \setminus \boldsymbol{Contains}(G_0, v_0))$ is empty. However, the formula $\mathbf{Conn}(G_0)$ holds and, since P_0 is also a subset of G_0, $\boldsymbol{Nodes}(P_0) \cap \boldsymbol{Nodes}(G_0 \setminus P_0)$ must be not empty by definition of $\mathbf{Conn}(\cdot)$. By writing $G_0 \setminus P_0$ as the union of $(G_0 \setminus P_0) \setminus \boldsymbol{Contains}(G_0, v_0)$ and $(G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0)$, we infer that $\boldsymbol{Nodes}(P_0) \cap (\boldsymbol{Nodes}((G_0 \setminus P_0) \setminus \boldsymbol{Contains}(G_0, v_0))) \cup \boldsymbol{Nodes}((G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0)))$ is nonnull by Lemma 10. Hence, the set $\boldsymbol{Nodes}((G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0)) \cap \boldsymbol{Nodes}(P_0) \neq \emptyset$, $\boldsymbol{Nodes}((G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0)) \neq \emptyset$, and, by the definition of $\boldsymbol{Nodes}(\cdot)$, $(G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0) \neq \emptyset$. Since $\boldsymbol{Contains}(G_0, v_0) = \{\{v_0, v_0'\}\}$, $\{v_0, v_0'\} \in G_0 \setminus P_0$ and $\{v_0, v_0'\} \notin P_0$. However, $P_0 \subseteq G_0$ and $\boldsymbol{Contains}(G_0, v_0) = \{e \in G_0 \mid v_0 \in e\} = \{\{v_0, v_0'\}\}$; therefore $\boldsymbol{Contains}(P_0, v_0) = \emptyset$, and $v_0 \notin \boldsymbol{Nodes}(P_0)$. Moreover, $(G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0)$ must be subset of $\{\{v_0, v_0'\}\}$ and, by Lemma 9, $\boldsymbol{Nodes}((G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0))$ is subset of $\boldsymbol{Nodes}\left(\{\{v_0, v_0'\}\}\right)$ which is $\{v_0, v_0'\}$ by the definition of $\boldsymbol{Nodes}(\cdot)$. The set $\boldsymbol{Nodes}((G_0 \setminus P_0) \cap \boldsymbol{Contains}(G_0, v_0)) \cap \boldsymbol{Nodes}(P_0)$ is nonnull; therefore it must equal $\{v_0'\}$, and v_0' belongs to $\boldsymbol{Nodes}(P_0)$. It follows that:

$$\begin{aligned}
&\boldsymbol{Nodes}(P_0 \cup \boldsymbol{Contains}(G_0, v_0)) = \\
&\qquad = \boldsymbol{Nodes}(P_0) \cup \boldsymbol{Nodes}(\boldsymbol{Contains}(G_0, v_0)) && \text{By Lemma 10} \\
&\qquad = \boldsymbol{Nodes}(P_0) \cup \boldsymbol{Nodes}\left(\{\{v_0, v_0'\}\}\right) && \text{By hypothesis} \\
&\qquad = \boldsymbol{Nodes}(P_0) \cup \{v_0, v_0'\} && \text{By definition of } \boldsymbol{Nodes}(\cdot) \\
&\qquad = \boldsymbol{Nodes}(P_0) \cup \{v_0\} && \text{Because } v_0 \notin \boldsymbol{Nodes}(P_0)
\end{aligned}$$

and, thus, $\textbf{\textit{Nodes}}\,(G_0 \setminus (P_0 \cup \textbf{\textit{Contains}}\,(G_0, v_0))) \cap \textbf{\textit{Nodes}}\,(P_0 \cup \textbf{\textit{Contains}}\,(G_0, v_0))$ is equal to both the sets $\textbf{\textit{Nodes}}\,((G_0 \setminus \textbf{\textit{Contains}}\,(G_0, v_0)) \setminus P_0)) \cap (\textbf{\textit{Nodes}}\,(P_0) \cup \{v_0\})$ and $\textbf{\textit{Nodes}}\,((G_0 \setminus \textbf{\textit{Contains}}\,(G_0, v_0)) \setminus P_0)) \cap \{v_0\}$. However, $G_0 \setminus \textbf{\textit{Contains}}\,(G_0, v_0)$ is equal to the set $\{e \in G_0 \mid v_0 \notin e\}$ by definition of $\textbf{\textit{Contains}}\,(\cdot)$. Hence, v_0 does not belong to $\textbf{\textit{Nodes}}\,(G_0 \setminus \textbf{\textit{Contains}}\,(G_0, v_0))$ by definition of $\textbf{\textit{Nodes}}\,(\cdot)$ and it does not also belong to $\textbf{\textit{Nodes}}\,(G_0 \setminus \textbf{\textit{Contains}}\,(G_0, v_0) \setminus P_0)$ by Lemma 9. Thus, the set $\textbf{\textit{Nodes}}\,(G_0 \setminus \textbf{\textit{Contains}}\,(G_0, v_0) \setminus P_0) \cap \{v_0\}$ is empty, so is the set $\textbf{\textit{Nodes}}\,(G_0 \setminus (\textbf{\textit{Contains}}\,(G_0, v_0) \cup P_0)) \cap \textbf{\textit{Nodes}}\,(P_0 \cup \textbf{\textit{Contains}}\,(G_0, v_0))$, and the formula $\textbf{Conn}\,(G_0)$ does not hold by definition of $\textbf{Conn}\,(\cdot)$. Since this last statement contradicts our assumptions, the claim must hold. □

Lemma 3 allows us to prove that whenever a vertex v belongs to a single edge e of G, G filtered w.r.t. v is still connected. As a matter of fact, $\textbf{\textit{Filter}}\,(G, v)$ contains all the set $e' \setminus \{v\}$ that have at least cardinality 2. Since v belongs only to e, if $e \setminus \{v\}$ is still an edge, i.e., it contains at least two nodes, the proof is straightforward because $\textbf{\textit{Nodes}}\,(G) \setminus \{v\} = \textbf{\textit{Nodes}}\,(\textbf{\textit{Filter}}\,(G, v))$ and, thus, because of the above remarks, $\textbf{\textit{Filter}}\,(G, v)$ is connected. If, instead, $e \setminus \{v\}$ is not an edge, then there exists a node v' of G such that $e = \{v, v'\}$ and $\textbf{\textit{Contains}}\,(G, v) = \{\{v, v'\}\}$. As a matter of fact, since G is connected either $G \setminus e$ is empty or v' belongs to another edge of G. In the former case, $G \setminus e$ is connected. In the latter, $\textbf{\textit{Nodes}}\,(G) \setminus \{v\} = \textbf{\textit{Nodes}}\,(G \setminus e)$ and, because of the above remarks, $G \setminus e = \textbf{\textit{Filter}}\,(G, v)$ must be connected.



(a) A graph G (b) $\textbf{\textit{Filter}}\,(G, b)$ (c) $\textbf{\textit{Filter}}\,(G, f)$

Fig. 4: If the set $\textbf{\textit{Contains}}\,(G, v)$ is a singleton and the hypergraph G is connected, the hypergraph $\textbf{\textit{Filter}}\,(G, v)$ is connected and v is a non-cut vertex.

Lemma 4 $(\textbf{\textit{Conn}}\,(G) \wedge \textbf{\textit{Contains}}\,(G, v) = \{e\}) \to \textbf{\textit{Conn}}\,(\textbf{\textit{Filter}}\,(G, v))$

Proof. By definition, the set $\textbf{\textit{Filter}}\,(G, v)$ is equal to $\{e' \setminus \{v\} \mid e' \in G \wedge |e' \setminus \{v\}|_{\geq 2}\}$. Since $\textbf{\textit{Contains}}\,(G, v) = \{e\}$, $\textbf{\textit{Filter}}\,(G, v) = (G \setminus \{e\}) \cup \{e \setminus \{v\} \mid |e \setminus \{v\}|_{\geq 2}\}$. There are two cases: $|e \setminus \{v\}|_{\geq 2}$ does not hold or it does. In the former case, since $|e|_{\geq 2}$ by definition of $\textbf{Graph}\,(\cdot)$, $v \in e$, there exists a v' such that $e = \{v, v'\}$, and $\textbf{\textit{Filter}}\,(G, v) = G \setminus \{e\}$. The claim follows from Lemma 3. If, otherwise, $|e \setminus \{v\}|_{\geq 2}$ holds, then $\textbf{\textit{Filter}}\,(G, v) = (G \setminus \{e\}) \cup \{e \setminus \{v\}\}$. Let us assume that there exist G_0, v_0, and e_0 such that both the formulæ $\textbf{Conn}\,(G_0) \wedge \textbf{\textit{Contains}}\,(G_0, v_0) = \{e_0\}$ and $\neg\textbf{Conn}\,(\textbf{\textit{Filter}}\,(G_0, v_0))$ hold. By definition of $\textbf{Conn}\,(\cdot)$, $\textbf{\textit{Nodes}}\,(G_0 \setminus P) \cap \textbf{\textit{Nodes}}\,(P) \neq \emptyset$ for all P such that $\emptyset \subsetneq P \subsetneq G_0$. Since $\neg\textbf{Conn}\,(\textbf{\textit{Filter}}\,(G_0, v_0))$

holds, there exists a P_0 such that $\emptyset \subsetneq P_0 \subsetneq (G_0 \setminus \{e_0\}) \cup \{e_0 \setminus \{v_0\}\} = \boldsymbol{Filter}(G_0, v_0)$ such that $\boldsymbol{Nodes}(P_0) \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0) \setminus P_0)$ is empty. Let be P_1 be one of P_0 and $\boldsymbol{Filter}(G_0, v_0) \setminus P_0$ such that $e_0 \setminus \{v_0\} \notin P_1$. Of course, $\boldsymbol{Nodes}(P_1) \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0) \setminus P_1) = \emptyset$ and $P_1 \subseteq \boldsymbol{Filter}(G_0, v_0) \setminus \{e_0 \setminus \{v_0\}\} \subseteq G_0$. The set $\boldsymbol{Filter}(G_0, v_0) \setminus P_1$ is equal to $((G_0 \setminus \{e_0\}) \setminus P_1) \cup \{e_0 \setminus \{v_0\}\}$ and to $((G_0 \setminus P_1) \setminus \{e_0\}) \cup \{e_0 \setminus \{v_0\}\}$. Thus, $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0) \setminus P_1)$ and $\boldsymbol{Nodes}(G_0 \setminus P_1) \setminus \{v_0\}$ are the same set by the definition of $\boldsymbol{Nodes}(\cdot)$ and $\boldsymbol{Nodes}(G_0 \setminus P_1) \cap \boldsymbol{Nodes}(P_1) \subseteq \{v_0\}$. However, v_0 does not belong to $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0))$, by definition of both $\boldsymbol{Nodes}(\cdot)$ and $\boldsymbol{Filter}(G_0, v_0)$, and $P_1 \subsetneq \boldsymbol{Filter}(G_0, v_0)$. Hence, by Lemma 9, $v_0 \notin \boldsymbol{Nodes}(P_1)$ and $\boldsymbol{Nodes}(G_0 \setminus P_1) \cap \boldsymbol{Nodes}(P_1) = \emptyset$. Thus, $\mathbf{Conn}(G_0)$ does not hold by definition of $\mathbf{Conn}(\cdot)$. This contradicts our assumptions and proves the claim. □

According to the definition of $\mathbf{Cutting}(\cdot)$, $\mathbf{Cutting}(G, v)$ holds if and only if G is connected, G contains at least two edges, and either there exists a node $v' \neq v$ of G that is not a node of $\boldsymbol{Filter}(G, v)$ or $\boldsymbol{Filter}(G, v)$ is not connected. If v is a boundary vertex and G is connected, $\boldsymbol{Filter}(G, v)$ must be connected too by Lemma 4. It follows that, under the above conditions, $\mathbf{Cutting}(G, v)$ holds if and only if G is connected, G contains at least two edges, and there exists a node $v' \neq v$ of G that is not a node of $\boldsymbol{Filter}(G, v)$. However, the latter condition holds if and only if $\boldsymbol{Contains}(G, v') = \{\{v, v'\}\}$ by Lemma 2. Thus, $\mathbf{Conn}(G \setminus \boldsymbol{Contains}(G, v))$ holds by Lemma 3 and $\mathbf{Conn}(\boldsymbol{Filter}(G, v))$. It follows that if v is a boundary vertex of G, then v is a non-cut edge.

Lemma 5 $\boldsymbol{Contains}(G, v) = \{e\} \rightarrow \neg \boldsymbol{Cutting}(G, v)$

Proof. Let us assume that there exist G_0, v_0, and e_0 such that $\boldsymbol{Contains}(G_0, v_0)$ is $\{e\}$ and the formula $\mathbf{Cutting}(G_0, v_0)$ holds. By definition of $\mathbf{Cutting}(\cdot)$, both the formulæ $\neg \mathbf{Conn}(\boldsymbol{Filter}(G_0, v_0)) \vee \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0)) \cup \{v_0\} \subsetneq \boldsymbol{Nodes}(G_0)$ and $\mathbf{Conn}(G_0) \wedge |G_0|_{\geq 2}$ hold. However, $\mathbf{Conn}(\boldsymbol{Filter}(G_0, v_0))$ must hold by Lemma 4. It follows that $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0)) \cup \{v_0\} \subsetneq \boldsymbol{Nodes}(G_0)$, there exists a $v_1 \in \boldsymbol{Nodes}(G_0) \setminus \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0)) \cup \{v_0\}$, and, by Lemma 2, $\boldsymbol{Contains}(G_0, v_1) = \{\{v_0, v_1\}\}$. Since $\boldsymbol{Contains}(G_0, v_0) = \{e_0\}$, from definition of $\boldsymbol{Contains}(\cdot)$ it follows that e_0 and $\{v_0, v_1\}$ are the same edge and, by Lemma 12, the set $\boldsymbol{Cov}(G_0, \{e_0\})$ is equal to the set $(\bigcup_{v \in \boldsymbol{Nodes}(e_0)} \boldsymbol{Contains}(G_0, v))$, which is $\{e_0\}$ by definition of $\boldsymbol{Nodes}(\cdot)$. By Lemma 13, $\boldsymbol{Nodes}(\{e_0\}) \cap \boldsymbol{Nodes}(G_0 \setminus \{e_0\})$ is empty and $\mathbf{Conn}(G_0)$ does not hold by definition of $\mathbf{Conn}(\cdot)$. Since this contradicts our assumptions, the claim must hold. □

As a direct consequence of Lemma 2 and Lemma 5, if G has a vertex $v' \neq v$ which is not a vertex of $\boldsymbol{Filter}(G, v)$, G has a non-cut vertex too.

Proof of Theorem 1. If $\boldsymbol{Nodes}(\boldsymbol{Filter}(G, v)) \cup \{v\}$ is a proper subset of $\boldsymbol{Nodes}(G)$, then the set $\boldsymbol{Lost}(G, v)$ is nonnull and it contains a vertex a v'. By Lemma 2, the set $\boldsymbol{Contains}(G, v')$ is equal to $\{\{v, v'\}\}$ and, by Lemma 5, $\neg \mathbf{Cutting}(G, v')$. □

3.2 Not all graphs $\textit{\textbf{Filter}}(G, v)$ are disconnected

This section proves that if G is connected, then it contains a node v such that $\textit{\textbf{Filter}}(G, v)$ is connected too.

Theorem 2 $\textit{\textbf{Conn}}(G) \to (G = \emptyset \vee \exists v \in \textit{\textbf{Nodes}}(G)\ \ \textit{\textbf{Conn}}(\textit{\textbf{Filter}}(G, v)))$

First of all, let us prove that, for any nonnull set P which is a proper subset of G and such that none of the nodes of P is a node of $\textit{\textbf{Filter}}(G, v) \setminus P$ and for any edge $e \in G$, either e does not contain any node of P or e does not contain any node of $\textit{\textbf{Filter}}(G, v) \setminus P$. Namely, if $\textit{\textbf{Filter}}(G, v)$ is partitioned into two disconnected subgraphs (i.e., P and $\textit{\textbf{Filter}}(G, v) \setminus P$), none of the edges of G "touches" both of these graphs. This is due to the fact that, besides vertices lost by v, which are boundary vertices by Lemma 2, only v belongs to $\textit{\textbf{Nodes}}(G)$ and not to $\textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v))$. Since $\textit{\textbf{Filter}}(G, v)$ is disconnected, both $e \setminus \textit{\textbf{Nodes}}(P)$ and $e \setminus \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v) \setminus P)$ are subsets of $\textit{\textbf{Lost}}(G, v) \cup \{v\}$ for each $e \in G$. Thus, either $e \cap \textit{\textbf{Nodes}}(P)$ or $e \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v) \setminus P)$ must be empty.

Lemma 6 $(\textit{\textbf{Graph}}(G) \wedge P \subseteq \textit{\textbf{Filter}}(G, v) \wedge \textit{\textbf{Nodes}}(P) \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v) \setminus P) = \emptyset) \to \forall e \in G(e \cap \textit{\textbf{Nodes}}(P) = \emptyset \vee e \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v) \setminus P) = \emptyset)$

Proof. Let us assume that there exist G_0, P_0, v_0, and e_0 such that both the formulæ $\mathbf{Graph}(G_0) \wedge P_0 \subseteq \textit{\textbf{Filter}}(G_0, v_0) \wedge \textit{\textbf{Nodes}}(P_0) \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0) = \emptyset$ and $e_0 \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0) \neq \emptyset \wedge e_0 \cap \textit{\textbf{Nodes}}(P_0) \neq \emptyset \wedge e_0 \in G_0$ hold. Either (a) $v_0 \notin e_0$ or (b) $v_0 \in e_0$ holds. In the former case $e_0 \in \textit{\textbf{Filter}}(G_0, v_0)$, by definition of $\textit{\textbf{Filter}}(\cdot)$ and $\mathbf{Graph}(\cdot)$, and we get a contradiction by Lemma 11. Hence, case (b) must hold. By our assumptions, there exist $v_1 \in e_0 \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0)$ and $v_2 \in e_0 \cap \textit{\textbf{Nodes}}(P_0)$. Since $\textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0) \cap \textit{\textbf{Nodes}}(P_0)$ is empty, $v_1 \neq v_2$ and $\{v_0, v_1, v_2\} \subseteq e_0$. Moreover, $v_0 \notin \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0))$ by definition of $\textit{\textbf{Filter}}(\cdot)$ and, since $P_0 \subseteq \textit{\textbf{Filter}}(G_0, v_0)$, $v_0 \notin \textit{\textbf{Nodes}}(P_0)$ and $v_0 \notin \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0)$ by Lemma 9. It follows that $v_0 \neq v_1$ and $v_0 \neq v_2$, $e_1 = e_0 \setminus \{v_0\}$ belongs to $\textit{\textbf{Filter}}(G_0, v_0)$ by definition of $\textit{\textbf{Filter}}(\cdot)$, and $|e_1|_{\geq 2}$. Since both $e_1 \cap \textit{\textbf{Nodes}}(P_0)$ and $e_1 \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0)$ are nonnull by construction and $P_0 \subseteq \textit{\textbf{Filter}}(G_0, v_0)$, $\textit{\textbf{Nodes}}(P_0) \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G_0, v_0) \setminus P_0)$ is nonnull by Lemma 11. This contradict our assumptions, hence the claim must hold. □

In the light of the above considerations, it is easy to see also that, if G is connected and $\textit{\textbf{Lost}}(G, v)$ is empty, then v is the only vertex in both of the graphs $\textit{\textbf{Cov}}(G, P)$ and $\textit{\textbf{Cov}}(G, \textit{\textbf{Filter}}(G, v) \setminus P)$.

Lemma 7 $(\textit{\textbf{Lost}}(G, v) = \emptyset \wedge \textit{\textbf{Conn}}(G) \wedge v \in \textit{\textbf{Nodes}}(G) \wedge$
$\emptyset \subsetneq P \subsetneq \textit{\textbf{Filter}}(G, v) \wedge \textit{\textbf{Nodes}}(P) \cap \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v) \setminus P) = \emptyset) \to$
$\textit{\textbf{Nodes}}(\textit{\textbf{Cov}}(G, P)) \cap \textit{\textbf{Nodes}}(\textit{\textbf{Cov}}(G, \textit{\textbf{Filter}}(G, v) \setminus P)) = \{v\}$

Proof. By the definition of $\mathbf{Conn}(\cdot)$, $\mathbf{Graph}(G)$ holds. By the definition of $\textit{\textbf{Lost}}(\cdot)$, if $\textit{\textbf{Lost}}(G, v) = \emptyset$, then $\textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v)) \supseteq \textit{\textbf{Nodes}}(G) \setminus \{v\}$. By Lemma 15, $\textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v)) \subseteq \textit{\textbf{Nodes}}(G) \setminus \{v\}$, hence $\textit{\textbf{Nodes}}(G) \setminus \{v\} = \textit{\textbf{Nodes}}(\textit{\textbf{Filter}}(G, v))$ and, by Lemma 16, $\textit{\textbf{Cov}}(G, P) \cup \textit{\textbf{Cov}}(G, \textit{\textbf{Filter}}(G, v) \setminus P) = G$. Since $\textit{\textbf{Nodes}}(P) \cap$

Nodes $(\boldsymbol{Filter}(G, v) \setminus P) = \emptyset$ for all nonnull P that is also a proper subset of ***Filter*** (G, v), ***Cov*** $(G, P) \cap$ ***Cov*** $(G, \boldsymbol{Filter}(G, v) \setminus P) = \emptyset$ by Lemma 17. As a consequence, ***Cov*** $(G, \boldsymbol{Filter}(G, v) \setminus P)$ is equal to $G \setminus \boldsymbol{Cov}(G, P)$ and ***Nodes*** $(\boldsymbol{Cov}(G, P)) \cap$ ***Nodes*** $(\boldsymbol{Cov}(G, \boldsymbol{Filter}(G, v) \setminus P)) \neq \emptyset$. By the definitions of ***Nodes*** $(\cdot)$ and ***Cov*** $(\cdot)$, v' belongs to ***Nodes*** $(\boldsymbol{Cov}(G, P)) \cap$ ***Nodes*** $(\boldsymbol{Cov}(G, \boldsymbol{Filter}(G, v) \setminus P))$ if and only if there exist $e, e' \in G$ such that $e \cap \boldsymbol{Nodes}(P)$ and $e' \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G, v) \setminus P)$ are nonnull and $v' \in e \cap e'$. Since $e, e' \in G$, $e \cap e'$ is equal to $(e \cap e') \cap \boldsymbol{Nodes}(G)$ by the definition of ***Nodes*** $(\cdot)$. Moreover $v \in \boldsymbol{Nodes}(G)$ by hypothesis, hence, ***Nodes*** (G) is equal to ***Nodes*** $(\boldsymbol{Filter}(G, v)) \cup \{v\}$ and to ***Nodes*** $(\boldsymbol{Filter}(G, v) \setminus P) \cup \boldsymbol{Nodes}(P) \cup \{v\}$ by Lemma 10. Thus, $v' \in e \cap e'$ if and only if either (a) $v' \in e \cap (e' \cap \boldsymbol{Nodes}(P))$, (b) $v' \in e' \cap (e \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G, v) \setminus P))$, or (c) $v' \in (e \cap e') \cap \{v\}$. However, due to Lemma 6, $e' \cap \boldsymbol{Nodes}(P)$ and $e \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G, v) \setminus P)$ must be empty and both cases (a) and (b) are not possible. It follows that the claim holds. □

By Lemma 2 and Lemma 4, if G is connected and ***Lost*** (G, v) is nonnull, then there exists a v' such that ***Filter*** (G, v') is connected. In order to prove Theorem 2 we are left to prove that, whenever G is connected and ***Lost*** (G, v) is empty, there exists a v' such that ***Filter*** (G, v') is connected.

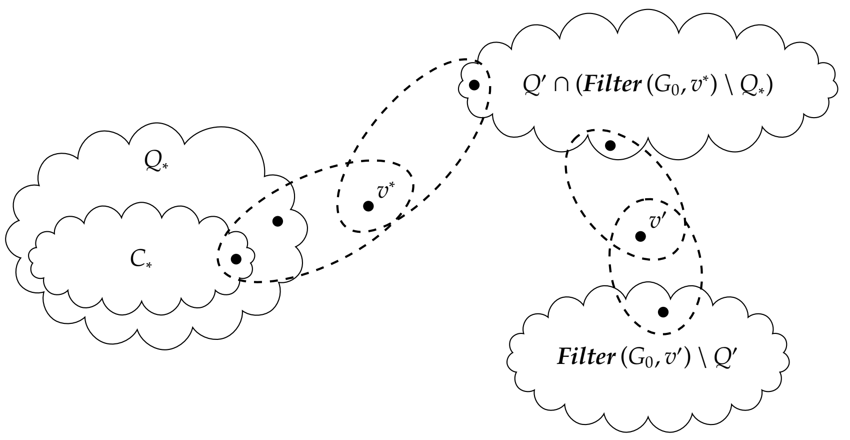

$Q' \cap (\boldsymbol{Filter}(G_0, v^*) \setminus Q_*)$
Q_*
v^*
C_*
v'
$\boldsymbol{Filter}(G_0, v') \setminus Q'$

Fig. 5: A graphical sketch of the proof of Theorem 2.

Let us assume by contradiction that the set ***Filter*** (G, v) is not connected for all $v \in \boldsymbol{Nodes}(G)$. From the finiteness of G, we can deduce that, for every vertex v of G, there exists a subset of ***Filter*** $(G, v) \cap G$ that is maximal and connected. Let us call it C_v and let $\boldsymbol{C}$ be the set of all these C_v's. Since G is finite, so is $\boldsymbol{C}$ and, then, there exists a graph C_* in $\boldsymbol{C}$ that is maximal. Let v^* be such that C_* is subset of ***Filter*** $(G, v^*) \cap G$. Since ***Filter*** (G, v^*) is disconnected, there exists a nonnull Q_* that is a proper subset of ***Filter*** (G, v^*) such that ***Nodes*** $(Q_*) \cap$ ***Nodes*** $(\boldsymbol{Filter}(G, v^*) \setminus Q_*)$ is empty, $C_* \subseteq Q_*$, and Q_* is minimal. However, none of the edges of G goes from Q_* to ***Filter*** $(G, v^*) \setminus Q_*$. Hence, for any node v' of

$\boldsymbol{Filter}(G, v^*) \setminus Q_*$ and for any edge e of $\boldsymbol{Contains}(G, v')$, e does not share any nodes with Q_*. Hence, $\boldsymbol{Cov}(G_0, C_*)$ is connected. Moreover, since $\boldsymbol{Filter}(G, v')$ is not connected either and $\boldsymbol{Contains}(G, v^*)$ must share some nodes with Q_*, there exists a nonnull Q' that is a proper subset of $\boldsymbol{Filter}(G, v')$ such that $\boldsymbol{Nodes}(Q') \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G, v') \setminus Q')$ is empty and $Q_* \subsetneq Q'$. Since $\boldsymbol{Contains}(G, v^*)$ must share some nodes with Q_* and Q_* is the minimal set that contains C_*, C_* must be a proper subset of $\boldsymbol{Cov}(G_0, C_*)$. This contradicts our assumptions and, thus, there must exist a $v \in \boldsymbol{Nodes}(G)$ such that $\boldsymbol{Filter}(G, v)$ is connected.

Proof of Theorem 2. As a preamble, notice that if $\mathbf{Conn}(G)$ and $\boldsymbol{Lost}(G, v) \neq \emptyset$ hold together, they imply, respectively, that G is a graph and that there is a $v' \in \boldsymbol{Lost}(G, v)$; hence $\boldsymbol{Contains}(G, v') = \{\{v, v'\}\}$ holds by Lemma 2, and $\mathbf{Conn}(\boldsymbol{Filter}(G, v'))$ holds by Lemma 4. Arguing by contradiction, suppose then that a G_0 exists satisfying $\mathbf{Conn}(G_0)$, $\forall v \in \boldsymbol{Nodes}(G_0) \neg\mathbf{Conn}(\boldsymbol{Filter}(G_0, v))$, $\forall v \in \boldsymbol{Nodes}(G_0)\, \boldsymbol{Lost}(G_0, v) = \emptyset$, and $G_0 \neq \emptyset$. Since $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v_0)) \subseteq \boldsymbol{Nodes}(G_0) \setminus \{v\}$ holds for all v, by Lemma 15, in view of the definition of $\boldsymbol{Lost}(\cdot)$ we get $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v)) = \boldsymbol{Nodes}(G_0) \setminus \{v\}$ for all v. Let C be the set $\{P \subseteq G_0 \mid \exists v \in \boldsymbol{Nodes}(G_0)\ P \subseteq \boldsymbol{Filter}(G_0, v) \wedge \mathbf{Conn}(P)\}$. From the finiteness of G_0, we get the finiteness of C and hence the existence of an inclusion-maximal $C_* \in C$; thus, for no $v \in \boldsymbol{Nodes}(G_0)$ there exists any $P \subseteq \boldsymbol{Filter}(G_0, v) \cap G_0$ such that $\mathbf{Conn}(P)$ and $C_* \subsetneq P$. The set C_* is empty if and only if $\boldsymbol{Filter}(G_0, v) \cap G_0$ is empty for all $v \in \boldsymbol{Nodes}(G_0)$, because $\mathbf{Conn}(\{e\})$ holds for all $e \in G_0$; accordingly, by Lemma 21, if $C_* = \emptyset$ then $\mathbf{Conn}(\boldsymbol{Filter}(G_0, v))$ holds for all $v \in \boldsymbol{Nodes}(G_0)$. However, this would contradict our assumptions; hence $C_* \neq \emptyset$ necessarily holds. It follows from $C_* \in C$ that a $v^* \in \boldsymbol{Nodes}(G_0)$ such that $C_* \subseteq \boldsymbol{Filter}(G_0, v^*)$ must exist. Moreover, since $\mathbf{Conn}(\boldsymbol{Filter}(G_0, v))$ does not hold for any $v \in \boldsymbol{Nodes}(G_0)$, a Q^* must exist such that $\emptyset \subsetneq Q^* \subsetneq \boldsymbol{Filter}(G_0, v^*)$ and $\boldsymbol{Nodes}(Q^*) \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v^*) \setminus Q^*) = \emptyset$ by the definition of $\mathbf{Conn}(\cdot)$. However, C_* is a subset of $\boldsymbol{Filter}(G_0, v^*)$ by construction; thus, by Lemma 18, either $C_* \subseteq Q^*$ or $C_* \subseteq \boldsymbol{Filter}(G_0, v^*) \setminus Q^*$ holds. Since $\boldsymbol{Filter}(G_0, v^*) \setminus (\boldsymbol{Filter}(G_0, v^*) \setminus Q^*)$ equals Q^* and $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v^*) \setminus Q^*) \cap \boldsymbol{Nodes}(Q^*) = \emptyset$ if and only if the set $\boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v^*) \setminus (\boldsymbol{Filter}(G_0, v^*) \setminus Q^*)) \cap \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v^*) \setminus Q^*)$ is empty, we can assume without loss of generality that $C_* \subseteq Q^*$. By the definition of $\boldsymbol{Filter}(\cdot)$, it holds that $v^* \notin \boldsymbol{Nodes}(\boldsymbol{Filter}(G_0, v^*))$ and, therefore, $v^* \notin \boldsymbol{Nodes}(Q^*)$. By Lemma 16, $G_0 = \boldsymbol{Cov}(G_0, Q^*) \cup \boldsymbol{Cov}(G_0, \boldsymbol{Filter}(G_0, v^*) \setminus Q^*)$, while $\boldsymbol{Cov}(G_0, \boldsymbol{Filter}(G_0, v^*) \setminus Q^*) \cap \boldsymbol{Cov}(G_0, Q^*) = \emptyset$ by Lemma 17. It follows that $\boldsymbol{Cov}(G_0, \boldsymbol{Filter}(G_0, v^*) \setminus Q^*)$ equals $G_0 \setminus \boldsymbol{Cov}(G_0, Q^*)$ and $\boldsymbol{Nodes}(\boldsymbol{Cov}(G_0, Q^*)) \cap \boldsymbol{Nodes}(\boldsymbol{Cov}(G_0, \boldsymbol{Filter}(G_0, v^*) \setminus Q^*)) = \{v^*\}$ by Lemma 7. Therefore we have $v^* \in \boldsymbol{Nodes}(\boldsymbol{Cov}(G_0, Q^*))$, whence $\mathbf{Conn}(\boldsymbol{Cov}(G_0, Q^*))$, by Lemma 19. Since $\emptyset \subsetneq C_* \subseteq Q^* \subsetneq \boldsymbol{Filter}(G_0, v^*)$, $C_* \subsetneq \boldsymbol{Cov}(G_0, Q^*)$ by Lemma 9, $\boldsymbol{Filter}(G_0, v^*) \setminus Q^*$ is nonnull, and so is $\boldsymbol{Cov}(G_0, \boldsymbol{Filter}(G_0, v^*) \setminus Q^*) = G_0 \setminus \boldsymbol{Cov}(G_0, Q^*)$ by Lemma 14. From $\mathbf{Graph}(G_0)$ it follows that there are no singletons in G_0. Hence, there exists a $v_1 \in \boldsymbol{Nodes}(G_0 \setminus \boldsymbol{Cov}(G_0, Q^*)) \setminus \{v^*\}$ by the definition of $\boldsymbol{Nodes}(\cdot)$. As a consequence of Lemma 20, we get $\boldsymbol{Cov}(G_0, Q^*) \subseteq \boldsymbol{Filter}(G_0, v_1)$, which contradicts the assumed maximality of C_{v^*} in C. This contradiction proves our claim. □

3.3 Every graph has a non-cut vertex

Corollary 1 is a a direct consequence of Theorem 1 and Theorem 2.

Proof of Corollary 1. Suppose that there exists a graph G_0 such that $\mathbf{Conn}(G_0)$, $G_0 \neq \emptyset$, and $\forall v \in \boldsymbol{Nodes}(G_0)$ $\mathbf{Cutting}(G_0, v)$ hold. By definition of $\mathbf{Cutting}(\cdot)$, the formula $\mathbf{Conn}(G_0) \wedge |G_0|_{\geq 2}$ holds and, for all $v \in \boldsymbol{Nodes}(G_0)$, either the set $\boldsymbol{Lost}(G_0, v)$ is nonnull or $\mathbf{Conn}(\boldsymbol{Filter}(G_0, v))$ does not hold. If $\boldsymbol{Lost}(G_0, v)$ is nonnull, then there exists a $v_0 \in \boldsymbol{Lost}(G_0, v)$ such that $\neg\mathbf{Cutting}(G_0, v_0)$ holds, by Theorem 1, and we get a contradiction. Hence, it must be the case that the formula $\mathbf{Conn}(G_0)$ holds, $\boldsymbol{Lost}(G_0, v)$ is empty, while $\mathbf{Conn}(\boldsymbol{Filter}(G_0, v))$ does not hold for any $v \in \boldsymbol{Nodes}(G_0)$. However, by Theorem 2, there must exists a $v' \in \boldsymbol{Nodes}(G_0)$ such that $\mathbf{Conn}(\boldsymbol{Filter}(G_0, v'))$ holds. This is the sought contradiction which proves our claim. □

4 Conclusions

This paper tackles the proof of a property of connected hypergraphs: every hypergraph has non-cut vertices. While this result is not new at all, the novelty of our work lies in the absence of the notion of path. As a matter of fact, we define as connected only those graphs whose edges cannot be parted into two nonnull subgraphs that do not share nodes. This approach has some relevance from a foundational point of view as it proves that paths are not necessary to identify non-cut vertices.

Since our proof is based on the ZF theory and it is complete down to the details, we plan to both verify its correctness by means of the proof checker Referee/Ætnanova [6] and to include these results in its scenarios on graphs (see, e.g., [5]). Documentation about these new experiments will be made available at http://aetnanova.units.it/scenarios/NonCutVertices/.

References

1. G.-B. Chae, E. M. Palmer, and R. W. Robinson. Counting labeled general cubic graphs. *Discrete Mathematics*, 307(23):2979–2992, 2007.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.
3. M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.
4. M. Milanič and A. I. Tomescu. Set Graphs. I. Hereditarily Finite Sets and Extensional Acyclic Orientations. *Discrete Applied Mathematics*, 161(4-5):677–690, 2013.
5. E. G. Omodeo and A. I. Tomescu. Set Graphs. III. Proof Pearl: Claw-Free Graphs Mirrored into Transitive Hereditarily Finite Sets. *Journal of Automated Reasoning*, 52(1):1–29, 2014.
6. J. T. Schwartz, D. Cantone, and E. G. Omodeo. *Computational Logic and Set Theory – Applying formalized Logic to Analysis*. Springer, 2011. Foreword by Martin Davis.
7. A. Tarski. Sur les ensembles finis. *Fundamenta Mathematicae*, 6(1):45–95, 1924.
8. F. Wiedijk. The QED Manifesto revisited. *Studies in Logic, Grammar and Rhetoric*, 10(23):121–133, 2007.

A Basic Properties

Lemma 8 $\exists e\ (e \in G \wedge v \in e) \leftrightarrow v \in \textbf{\textit{Nodes}}(G)$

Proof. ($\leftarrow$) By the definition of $\textbf{\textit{Nodes}}(\cdot)$, $\textbf{\textit{Nodes}}(G) = \bigcup_{e \in G} e$. Hence, if $v \in \textbf{\textit{Nodes}}(G)$, there should exists a $e_0 \in G$ such that $v \in e_0$.

($\rightarrow$) By the definition of $\textbf{\textit{Nodes}}(\cdot)$, $\textbf{\textit{Nodes}}(G) = \bigcup_{e \in G} e$. Hence, if there exists a e_0 such that $e_0 \in G \wedge v \in e_0$, then $v \in \textbf{\textit{Nodes}}(G)$. □

Lemma 9 $P \subseteq G \rightarrow \textbf{\textit{Nodes}}(P) \subseteq \textbf{\textit{Nodes}}(G)$

Proof. By Lemma 8, if $v \in \textbf{\textit{Nodes}}(P)$, then there exists $e \in P$ such that $v \in e$. Since $P \subseteq G$ by hypothesis, $e \in G$ and, by Lemma 8, $v \in \textbf{\textit{Nodes}}(G)$. □

Lemma 10 $\textbf{\textit{Nodes}}(P \cup Q) = \textbf{\textit{Nodes}}(P) \cup \textbf{\textit{Nodes}}(Q)$

Proof. ($\supseteq$) Since $P \subseteq P \cup Q$, by Lemma 9, $\textbf{\textit{Nodes}}(P) \subseteq \textbf{\textit{Nodes}}(P \cup Q)$. Analogously, we get that $\textbf{\textit{Nodes}}(Q) \subseteq \textbf{\textit{Nodes}}(P \cup Q)$. Hence, $\textbf{\textit{Nodes}}(Q) \cup \textbf{\textit{Nodes}}(P) \subseteq \textbf{\textit{Nodes}}(P \cup Q) \cup \textbf{\textit{Nodes}}(P \cup Q) = \textbf{\textit{Nodes}}(P \cup Q)$.

($\subseteq$) By Lemma 8, if $v \in \textbf{\textit{Nodes}}(P \cup Q)$ then there exists a $e \in P \cup Q$ such that $v \in e$. Hence, e belongs to either P or Q. If $e \in P$, then $v \in \textbf{\textit{Nodes}}(P)$ by Lemma 8. Symmetrically, if $e \in Q$, then $v \in \textbf{\textit{Nodes}}(Q)$ by Lemma 8. Thus, if $v \in \textbf{\textit{Nodes}}(P \cup Q)$, then $v \in \textbf{\textit{Nodes}}(P) \cup \textbf{\textit{Nodes}}(Q)$. □

Lemma 11 $P \subseteq G \rightarrow (\textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(G \setminus P) = \emptyset \leftrightarrow$
$\forall e \in G(e \cap \textbf{\textit{Nodes}}(P) = \emptyset \vee e \cap \textbf{\textit{Nodes}}(G \setminus P) = \emptyset))$

Proof. By the definition of $\textbf{\textit{Nodes}}(\cdot)$, $\textbf{\textit{Nodes}}(G) = \bigcup_{e \in G} e$. Thus, $\textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(G \setminus P) = \emptyset$ if and only if both $e \cap \textbf{\textit{Nodes}}(P)$ and $e' \cap \textbf{\textit{Nodes}}(G \setminus P)$ are empty for all $e \in G \setminus P$ and for all $e' \in P$. However, if $P \subseteq G$, then $(G \setminus P) \cup P$ is equal to G. Thus, if $P \subseteq G$, then $\textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(G \setminus P) = \emptyset$ if and only if $\forall e \in G\ (e \cap \textbf{\textit{Nodes}}(P) = \emptyset \vee e \cap \textbf{\textit{Nodes}}(G \setminus P) = \emptyset)$. □

Lemma 12 $\textbf{\textit{Cov}}(G, P) = \bigcup_{v \in \textbf{\textit{Nodes}}(P)} \textbf{\textit{Contains}}(G, v)$

Proof. We prove that $e \in \textbf{\textit{Cov}}(G, P)$ if and only if $e \in \bigcup_{v \in \textbf{\textit{Nodes}}(P)} \textbf{\textit{Contains}}(G, v)$. By definition of $\textbf{\textit{Cov}}(\cdot)$, $e \in \textbf{\textit{Cov}}(G, P)$ if and only if and only if $e \in G$ and $e \cap \textbf{\textit{Nodes}}(P) \neq \emptyset$. Hence, $e \in \textbf{\textit{Cov}}(G, P)$ if and only if there exists a $v \in \textbf{\textit{Nodes}}(P)$ and a $e \in G$ such that $v \in e$. By definition of $\textbf{\textit{Contains}}(\cdot)$, $e \in \textbf{\textit{Contains}}(G, v)$ if and only if $v \in e$ and $e \in G$. Thus, $e \in \textbf{\textit{Cov}}(G, P)$ if and only if there exists a $v \in \textbf{\textit{Nodes}}(P)$ such that $e \in \textbf{\textit{Contains}}(G, v)$. The thesis follows directly. □

Lemma 13 $\textbf{\textit{Cov}}(G, P) \subseteq P \leftrightarrow \textbf{\textit{Nodes}}(G \setminus P) \cap \textbf{\textit{Nodes}}(P) = \emptyset$

Proof. By the definition of $\textbf{\textit{Cov}}(\cdot)$, $\textbf{\textit{Cov}}(G, P)$ is equal to $\{e \in G | e \cap \textbf{\textit{Nodes}}(P) \neq \emptyset\}$. Hence, $\textbf{\textit{Cov}}(G, P) \subseteq P$ if and only if $e \cap \textbf{\textit{Nodes}}(P) = \emptyset$ for all $e \in G \setminus P$ and if and only if $(\bigcup_{e \in G \setminus P} e) \cap \textbf{\textit{Nodes}}(P) = \emptyset$. By definition of $\textbf{\textit{Nodes}}(\cdot)$, this is equivalent to $\textbf{\textit{Nodes}}(G \setminus P) \cap \textbf{\textit{Nodes}}(P) = \emptyset$ and, thus, the thesis holds. □

Lemma 14 $\textbf{\textit{Nodes}}(P) \subseteq \textbf{\textit{Nodes}}(G) \wedge \textbf{\textit{Nodes}}(P) \neq \emptyset \to \textbf{\textit{Cov}}(G,P) \neq \emptyset$

Proof. Let us assume that there exist G_0 and P_0 such that both $\textbf{\textit{Nodes}}(P_0) \subseteq \textbf{\textit{Nodes}}(G_0) \wedge \textbf{\textit{Nodes}}(P_0) \neq \emptyset$ and $\textbf{\textit{Cov}}(G_0,P_0) = \emptyset$ hold. Since $\textbf{\textit{Cov}}(G_0,P_0)$ is equal to $\{e \in G_0 \mid e \cap \textbf{\textit{Nodes}}(P_0) \neq \emptyset\}$ by the definition of $\textbf{\textit{Cov}}(\cdot)$, $\textbf{\textit{Cov}}(G_0,P_0) = \emptyset$ if and only if $\textbf{\textit{Nodes}}(G_0) \cap \textbf{\textit{Nodes}}(P_0)$ is empty. However, from $\textbf{\textit{Nodes}}(P_0) \subseteq \textbf{\textit{Nodes}}(G_0)$, we deduce that $\textbf{\textit{Nodes}}(G_0) \cap \textbf{\textit{Nodes}}(P_0)$ and $\textbf{\textit{Nodes}}(P_0)$ are the same set and, from $\textbf{\textit{Nodes}}(P_0) \neq \emptyset$, we get that $\textbf{\textit{Nodes}}(G_0) \cap \textbf{\textit{Nodes}}(P_0)$ is not empty. This leads to a contradiction and proves our goal. □

Lemma 15 $\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) \subseteq \textbf{\textit{Nodes}}(G) \setminus \{v\}$

Proof. By the definition of $\textbf{\textit{Nodes}}(\cdot)$, if $\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) = \bigcup_{e' \in \textbf{\textit{Filter}}(G,v)} e'$. However, by the definition of $\textbf{\textit{Filter}}(\cdot)$, $e' \in \textbf{\textit{Filter}}(G,v)$ if any only if there exists a $e \in G$ such that $e' = e \setminus \{v\}$ and $|e \setminus \{v\}|_{\geq 2}$. Thus, $\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) = \bigcup_{e \in G \mid |e \setminus \{v\}|_{\geq 2}} e \setminus V$. Since $\{e \in G \mid |e \setminus \{v\}|_{\geq 2}\} \subseteq G$, $\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,V)) \subseteq \bigcup_{e \in G} e \setminus \{v\} = (\bigcup_{e \in G} e) \setminus \{v\}$. Hence, by the definition of $\textbf{\textit{Nodes}}(\cdot)$, $\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) \subseteq \textbf{\textit{Nodes}}(G) \setminus \{v\}$. □

B Auxiliary Lemmas

Lemma 16 $(\textbf{\textit{Graph}}(G) \wedge \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v)) = \textbf{\textit{Nodes}}(G) \setminus \{v\} \wedge$
$P \subseteq \textbf{\textit{Filter}}(G,v)) \to \textbf{\textit{Cov}}(G,P) \cup \textbf{\textit{Cov}}(G,\textbf{\textit{Filter}}(G,v) \setminus P) = G$

Proof. Let us assume that there exist G_0, P_0, and v_0 such that both the formulæ $\textbf{Graph}(G_0) \wedge \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G_0,v_0)) = \textbf{\textit{Nodes}}(G_0) \setminus \{v_0\} \wedge P_0 \subseteq \textbf{\textit{Filter}}(G_0,v_0)$ $\textbf{\textit{Cov}}(G_0,P_0) \cup \textbf{\textit{Cov}}(G_0,\textbf{\textit{Filter}}(G_0,v_0) \setminus P_0) \neq G_0$ hold. By the definition of $\textbf{\textit{Cov}}(\cdot)$, $\textbf{\textit{Cov}}(G_0,P)$ is the set $\{e \in G_0 \mid e \cap \textbf{\textit{Nodes}}(P) \neq \emptyset\}$. Thus, $\textbf{\textit{Cov}}(G_0,P) \subseteq G_0$ and $\textbf{\textit{Cov}}(G_0,P_0) \cup \textbf{\textit{Cov}}(G_0,\textbf{\textit{Filter}}(G_0,v_0) \setminus P_0) \subseteq G_0$. However, $\textbf{\textit{Cov}}(G_0,P_0) \cup \textbf{\textit{Cov}}(G_0,\textbf{\textit{Filter}}(G_0,v_0) \setminus P_0) \neq G_0$ by assumption, hence, there exists a $e_0 \in G_0$ that belongs to neither $\textbf{\textit{Cov}}(G_0,P_0)$ nor $\textbf{\textit{Cov}}(G_0,\textbf{\textit{Filter}}(G_0,v_0) \setminus P_0)$. By the definition of $\textbf{\textit{Cov}}(\cdot)$, $e_0 \cap \textbf{\textit{Nodes}}(P_0) = \emptyset$ and $e_0 \cap \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G_0,v_0) \setminus P_0) = \emptyset$. Thus $e_0 \cap (\textbf{\textit{Nodes}}(P_0) \cup \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G_0,v_0) \setminus P_0)) = \emptyset$ and $e_0 \cap \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G_0,v_0))$ is empty by Lemma 10. Since $\textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G_0,v_0)) = \textbf{\textit{Nodes}}(G_0) \setminus \{v_0\}$ by assumption, $e_0 \cap (\textbf{\textit{Nodes}}(G_0) \setminus \{v_0\})$ is empty and $e_0 \cap \textbf{\textit{Nodes}}(G_0) \subseteq \{v_0\}$. By definition of $\textbf{\textit{Nodes}}(\cdot)$, $e_0 \subseteq \textbf{\textit{Nodes}}(G_0)$. Thus, $e_0 \cap \textbf{\textit{Nodes}}(G_0) = e_0$ and $e_0 \subseteq \{v_0\}$. However, this contradicts $|e_0|_{\geq 2}$ which must hold because of the definition of $\textbf{Graph}(\cdot)$. The claim of this lemma follows readily. □

Lemma 17 $(\textbf{\textit{Graph}}(G) \wedge \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v) \setminus P) \cap \textbf{\textit{Nodes}}(P) = \emptyset \wedge$
$P \subseteq \textbf{\textit{Filter}}(G,v)) \to \textbf{\textit{Cov}}(G,P) \cap \textbf{\textit{Cov}}(G,\textbf{\textit{Filter}}(G,v) \setminus P) = \emptyset$

Proof. By Lemma 6, either $e \cap \textbf{\textit{Nodes}}(P)$ or $e \cap \textbf{\textit{Nodes}}(\textbf{\textit{Filter}}(G,v) \setminus P)$ are empty for all $e \in G$. By the definition of $\textbf{\textit{Cov}}(\cdot)$, it follows that either $e \notin \textbf{\textit{Cov}}(G,P)$ or $e \notin \textbf{\textit{Cov}}(G,\textbf{\textit{Filter}}(G,v) \setminus P)$ for all $e \in G$. Thus, $\textbf{\textit{Cov}}(G,P) \cap \textbf{\textit{Cov}}(G,\textbf{\textit{Filter}}(G,v) \setminus P)$ is empty and the claim holds. □

Lemma 18 $(\textbf{\textit{Nodes}}(Q) \cap \textbf{\textit{Nodes}}(G \setminus Q) = \emptyset \wedge P \subseteq G \wedge \textbf{\textit{Conn}}(P)) \rightarrow$
$(P \subseteq G \setminus Q \vee P \subseteq Q)$

Proof. Let us assume that there exist G_0, P_0, and Q_0 such that both the formulæ $\textbf{\textit{Nodes}}(Q_0) \cap \textbf{\textit{Nodes}}(G_0 \setminus Q_0) = \emptyset \wedge P_0 \subseteq G_0 \wedge \textbf{Conn}(P_0)$ and $P_0 \not\subseteq G_0 \setminus Q_0 \wedge P_0 \not\subseteq Q_0$ hold. Hence, there exist an $e_0 \in P_0$ that does not belong to $G_0 \setminus Q_0$ and an $e_1 \in P_0$ that does not belong to Q_0. It follows that both the sets $P_0 \setminus Q_0$ and $P_0 \setminus (G_0 \setminus Q_0)$ are nonnull. Since $P_0 \subseteq G_0$, e_0 belongs to Q_0. Thus, $P_0 \setminus Q_0$ is a nonnull set that is also proper subset of P_0. By definition of $\textbf{Conn}(\cdot)$, $\textbf{Conn}(P_0)$ holds if and only if the set $\textbf{\textit{Nodes}}(P_0 \setminus S) \cap \textbf{\textit{Nodes}}(S)$ is nonnull for all not empty S that are also proper subsets of P_0. In particular, the set $\textbf{\textit{Nodes}}(P_0 \setminus (P_0 \setminus Q_0)) \cap \textbf{\textit{Nodes}}(P_0 \setminus Q_0)$ is nonnull. Thus,

$$\begin{aligned}
\emptyset \subsetneq & \textbf{\textit{Nodes}}(P_0 \setminus (P_0 \setminus Q_0)) \cap \textbf{\textit{Nodes}}(P_0 \setminus Q_0) & \\
= & \textbf{\textit{Nodes}}(P_0 \cap Q_0) \cap \textbf{\textit{Nodes}}(P_0 \setminus Q_0) & \\
\subseteq & \textbf{\textit{Nodes}}(Q_0) \cap \textbf{\textit{Nodes}}(P_0 \setminus Q_0) & \text{Since } P_0 \cap Q_0 \subseteq P_0 \text{, by Lemma 9} \\
\subseteq & \textbf{\textit{Nodes}}(Q_0) \cap \textbf{\textit{Nodes}}(G_0 \setminus Q_0) & \text{Since } P_0 \subseteq G_0 \text{, by Lemma 9}
\end{aligned}$$

This contradicts our assumptions and proves the claim. □

Lemma 19 $(\textbf{\textit{Conn}}(P \cup Q) \wedge \textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(Q) = \{v\}) \rightarrow \textbf{\textit{Conn}}(P)$

Proof. Let us assume that there exist P_0, Q_0 and v_0 such that both the formulæ $\textbf{Conn}(P_0 \cup Q_0) \wedge \textbf{\textit{Nodes}}(P_0) \cap \textbf{\textit{Nodes}}(Q_0) = \{v_0\}$ and $\neg\textbf{Conn}(P_0)$ hold. By definition of $\textbf{Conn}(\cdot)$, there exists a P_1 such that $\emptyset \subsetneq P_1 \subsetneq P_0$ and $\textbf{\textit{Nodes}}(P_1) \cap \textbf{\textit{Nodes}}(P_0 \setminus P_1)$ is empty. Furthermore,

$$\begin{aligned}
\{v_0\} = & \textbf{\textit{Nodes}}(P_0) \cap \textbf{\textit{Nodes}}(Q_0) & \text{By assumption} \\
= & \textbf{\textit{Nodes}}((P_0 \setminus P_1) \cup P_1) \cap \textbf{\textit{Nodes}}(Q_0) & \text{Since } P_1 \subseteq P_0 \\
= & (\textbf{\textit{Nodes}}(P_0 \setminus P_1) \cup \textbf{\textit{Nodes}}(P_1)) \cap \textbf{\textit{Nodes}}(Q_0) & \text{By Lemma 10} \\
= & (\textbf{\textit{Nodes}}(P_0 \setminus P_1) \cap \textbf{\textit{Nodes}}(Q_0)) \cup & \\
& \qquad (\textbf{\textit{Nodes}}(P_1) \cap \textbf{\textit{Nodes}}(Q_0)) &
\end{aligned}$$

Thus, either $v_0 \in \textbf{\textit{Nodes}}(P_0 \setminus P_1) \cap \textbf{\textit{Nodes}}(Q_0)$ or $v_0 \in \textbf{\textit{Nodes}}(P_1) \cap \textbf{\textit{Nodes}}(Q_0)$. Since $\textbf{\textit{Nodes}}(P_0 \setminus P_1) \cap \textbf{\textit{Nodes}}(P_1) = \emptyset$, either $v_0 \notin \textbf{\textit{Nodes}}(P_0 \setminus P_1)$ or $v_0 \notin \textbf{\textit{Nodes}}(P_1)$. Moreover, $(\textbf{\textit{Nodes}}(P_0 \setminus P_1) \cap \textbf{\textit{Nodes}}(Q_0)) \cup (\textbf{\textit{Nodes}}(P_1) \cap \textbf{\textit{Nodes}}(Q_0))$ is equal to $\{v_0\}$, hence, one of the two sets $\textbf{\textit{Nodes}}(P_0 \setminus P_1) \cap \textbf{\textit{Nodes}}(Q_0)$ and $\textbf{\textit{Nodes}}(P_1) \cap \textbf{\textit{Nodes}}(Q_0)$ is empty. Let be P_2 be such that $P_2 \in \{P_1, P_0 \setminus P_1\}$ and $\textbf{\textit{Nodes}}(P_2) \cap \textbf{\textit{Nodes}}(Q_0)$ is empty. Since $\emptyset \subsetneq P_1 \subsetneq P_0$, P_2 is nonnull, it is a proper subset of P_0 and $(P_0 \setminus P_2) \cup P_2 = P_0$. Moreover, $P_1 = P_0 \setminus (P_0 \setminus P_1)$ and, hence, $\textbf{\textit{Nodes}}(P_2) \cap \textbf{\textit{Nodes}}(P_0 \setminus P_2) = \textbf{\textit{Nodes}}(P_1) \cap \textbf{\textit{Nodes}}(P_0 \setminus P_1) = \emptyset$ is empty. By the definition of $\textbf{\textit{Nodes}}(\cdot)$, $\textbf{\textit{Nodes}}(P_0) \cap \textbf{\textit{Nodes}}(Q_0) = \{v_0\}$ if and only if $e \cap e' \subseteq \{v_0\}$ for all $e \in P_0$ and for all $e' \in Q_0$. Hence, $|e \cap e'|_{\geq 2}$ does not hold and, by the definition of $\textbf{Graph}(\cdot)$, this means that $P_0 \cap Q_0 = \emptyset$. Since $P_2 \subsetneq P_0$, $P_2 \cap Q_0 \subseteq P_0 \cap Q_0$

and $P_2 \cap Q_0$ is empty. It follows that

$$\begin{aligned}
\emptyset =& (\textbf{\textit{Nodes}}(P_2) \cap \textbf{\textit{Nodes}}(P_0 \setminus P_2)) \cup \\
& \qquad (\textbf{\textit{Nodes}}(P_2) \cap \textbf{\textit{Nodes}}(Q_0)) \\
=& \textbf{\textit{Nodes}}(P_2) \cap (\textbf{\textit{Nodes}}(P_0 \setminus P_2) \cup \textbf{\textit{Nodes}}(Q_0)) \\
=& \textbf{\textit{Nodes}}(P_2) \cap \textbf{\textit{Nodes}}((P_0 \setminus P_2) \cup Q_0) && \text{By Lemma 10} \\
=& \textbf{\textit{Nodes}}(P_2) \cap \textbf{\textit{Nodes}}((P_0 \cup Q_0) \setminus P_2) && \text{Since } Q_0 \cap P_2 = \emptyset
\end{aligned}$$

By the definition of **Conn** $(\cdot)$, **Conn** $(P_0 \cup Q_0)$ does not hold. This contradicts our assumptions and prove the claim. □

Lemma 20 $(\textbf{\textit{Graph}}(P \cup Q) \wedge \textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(Q) = \{v\}) \rightarrow$
$\forall v' \in \textbf{\textit{Nodes}}(Q) \setminus \{v\}\ (P \subseteq \textbf{\textit{Filter}}(P \cup Q, v'))$

Proof. By definition of ***Filter*** $(\cdot)$,

$$\begin{aligned}
\textbf{\textit{Filter}}(P \cup Q, v') =& \{e \setminus \{v'\} \mid e \in (P \cup Q) \wedge |e \setminus \{v'\}|_{\geq 2}\} \\
=& \{e \setminus \{v'\} \mid e \in P \wedge |e \setminus \{v'\}|_{\geq 2}\} \cup \{e \setminus \{v'\} \mid e \in Q \wedge |e \setminus \{v'\}|_{\geq 2}\}
\end{aligned}$$

Since $v' \in \textbf{\textit{Nodes}}(Q) \setminus \{v\}$ and $\textbf{\textit{Nodes}}(P) \cap \textbf{\textit{Nodes}}(Q) = \{v\}$, then $v' \notin \textbf{\textit{Nodes}}(P)$. By definition of ***Nodes*** $(\cdot)$, it follows that $v' \notin e$ and $e = e \setminus \{v'\}$ for any $e \in P$. Thus,

$$\begin{aligned}
\textbf{\textit{Filter}}(P \cup Q, v') =& \{e \mid e \in P \wedge |e|_{\geq 2}\} \cup \{e \setminus \{v'\} \mid e \in Q \wedge |e \setminus \{v'\}|_{\geq 2}\} \\
=& P \cup \{e \setminus \{v'\} \mid e \in Q \wedge |e \setminus \{v'\}|_{\geq 2}\}
\end{aligned}$$

The claim readily follows from the last equation. □

Lemma 21 $(\textbf{\textit{Graph}}(G) \wedge \forall v \in \textbf{\textit{Nodes}}(G)\ \textbf{\textit{Filter}}(G, v) \cap G = \emptyset) \rightarrow$
$\forall v \in \textbf{\textit{Nodes}}(G)\ \textbf{\textit{Conn}}(\textbf{\textit{Filter}}(G, v))$

Proof. The graph $\textbf{\textit{Filter}}(G, v) \cap G$ is empty for all $v \in \textbf{\textit{Nodes}}(G)$ if and only if $\textbf{\textit{Contains}}(G, v) = G$ for all $v \in \textbf{\textit{Nodes}}(G)$, i.e., every edge of G contains all nodes of G. It follows that G is a singleton and that each $\textbf{\textit{Filter}}(G, v)$ is either empty or a singleton; hence, by the definition of **Conn** $(\cdot)$, **Conn** $(\textbf{\textit{Filter}}(G, v))$ holds for all $v \in \textbf{\textit{Nodes}}(G)$. The claim follows readily. □

Binary 3-compressible automata

Alessandra Cherubini[1]* and Andrzej Kisielewicz[2]**

[1] Politecnico di Milano, Dipartimento di Matematica
[2] Department of Mathematics and Computer Science, University of Wrocław
alessandra.cherubini@polimi.it, andrzej.kisielewicz@math.uni.wroc.pl

Abstract. A finite deterministic automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is k-compressible if there is a word $w \in \Sigma^+$ such that the image of the state set Q under the natural action of w is reduced by at least k states. In such case w is called a k-compressing word for $\mathcal{A}$. It is known that, for any alphabet Σ and any $k \geq 2$, there exist words that are k-compressing for each k-compressible automaton with the input alphabet Σ. Such words are called k-collapsing. It has been proved that recognizing 2-collapsing words over a 2-element alphabet may be done in polynomial time, while recognizing 2-collapsing words over an alphabet of size ≥ 3 is co-NP-complete. A natural question in this context, whether recognizing 3-collapsing words over a 2-element alphabet is easy or hard, has remained open. In this paper we provide results on 3-compressible binary automata, which allow to prove that that the latter problem is co-NP-complete.

1 Introduction

Let $T(Q)$ be the full transformation monoid on a finite set Q. For $f \in T(Q)$, we define the *deficiency* $df(f)$ of f to be the difference between the cardinalities of Q and the image Imf under f, $df(f) = |Q| - |Imf|$. At the beginning of 1990's Sauer and Stone [12] introduced the property Δ_k defined as follows. For a finite alphabet Σ and a positive integer k, a word $w \in \Sigma^+$ has the property Δ_k for Σ if for all homomorphisms $\phi : \Sigma^+ \to T(Q)$, where Q is any finite set, $df(w\phi) \geq k$ whenever $df(v\phi) \geq k$ for some $v \in \Sigma^+$. They proved the nonobvious fact that such words exist for each positive integer k and for each finite alphabet Σ giving an elegant recursive construction that produces a word whose length is $O(2^{2^k})$. Their construction was improved in [9], where better but yet unrealistic upper bounds for the length of shortest words with the property Δ_k were given.

Words with the property Δ_k have a natural interpretation in the language of finite automata theory. Each complete deterministic automaton $\mathcal{A} = (Q, \Sigma, \delta)$ over the alphabet Σ can be viewed as the transformation monoid generated by transformations on Q induced via δ by the letters of Σ. Namely, for each $\alpha \in \Sigma$ we define the induced transformation by $q\alpha = \delta(q, \alpha)$. This action of the letters

* Supported in part by PRIN: "Automi e linguaggi formali: aspetti matematici e applicativi".
** Supported in part by Polish NCN grant 2012/07/B/ST1/03318.

on Q extends naturally into the action of words $w \in \Sigma^+$ on Q which is denoted briefly by $qw = \delta(q, \alpha)$.

Conversely, to define an automaton it is enough to assign to any letter of Σ a transformation on Q. Thus an automaton $\mathcal{A}$ can be identified with a specific homomorphism $\phi_{\mathcal{A}}$ of Σ^+ in $T(Q)$. If there exists a word $v \in \Sigma^+$ such that $df(v\phi_{\mathcal{A}}) \geq k$—that is, if $|Q| - |Qv| \geq k$—the automaton $\mathcal{A}$ is called *k-compressible* and v is a *k-compressing* word for $\mathcal{A}$ (it *k-compresses* $\mathcal{A}$). A word that is k-compressing for each k-compressible automaton with the input alphabet Σ is called *k-collapsing.* Obviously, a word has the property Δ_k (or witnesses for deficiency k, in the terminology of [9]) if and only if it is k-collapsing. Hence, besides the original motivations coming from combinatorics and algebra, the interest in such words comes from the fact that they can be seen as universal testers whose action on the set of states of an automaton exposes whether or not the automaton is k-compressible. The problem of the length of the shortest k-collapsing word over an alphabet Σ can be considered as a black-box version of the generalized Černý's conjecture stated by Pin [7, 8].

In [10] it is proved that the membership of a given word $w \in \Sigma^+$ to the language of k-collapsing words is decidable. The decision procedure is in the class co-NP and requires linear space, which shows that the language of k-collapsing words is context-sensitive. In [11] it is shown that it is not context-free even in the very simple case of the language of 2-collapsing words over a 2-letter alphabet. Most results so far concern 2-collapsing words. In particular, 2-collapsing words were characterized in [2] and in [5]. From the first characterization, that has a group theoretical flavor, a non-deterministic polynomial algorithm to recognize whether a word $w \in \Sigma^+$ is 2-collapsing was derived [3]. A refinement of this algorithm was used in [4] to give the list of shortest 2-collapsing words over a 3-letter alphabet. The second characterization is in terms of systems of permutation conditions and it is used in [6] to show that the membership problem of the language of 2-collapsing words over an alphabet of size ≥ 3 is a co-NP-complete problem. The algorithms for recognizing whether a word $w \in \Sigma^+$ is 2-collapsing, derived by both the characterizations of 2-collapsing words, become polynomial algorithms when $|\Sigma| = 2$ even if w is represented in the compressed form [5]. In view of these results a question arose whether for $k \geq 3$, k-collapsing words over a binary alphabet can also be recognized in polynomial time. This natural question has remained open so far. In this paper we show that the answer is negative. We prove that the problem of recognizing whether a word $w \in \{\alpha, \beta\}^+$ is 3-collapsing is co-NP-complete.

The difficulty in this case is that we have no characterization of 3-collapsing words similar to that for the case of 2-collapsing words. Yet, we have a certain classification of proper 3-compressible automata on 2 input letters [1], and we can see that the problem looks differently depending on the class. In some classes it is easy to recognize whether a word w 3-compresses all the automata in the class. There is however at least one class where this problem leads to solving a sort of a system of permutation conditions, similarly as in [6], and it seems computationally hard.

Our idea is the following. First, we reduce the problem whether a word w is 3-collapsing to the problem of recognizing those words w that 3-compress all automata in a restricted class $\mathcal{D}$ of 3-compressible automata. Next, we show that in this restricted class the problem is equivalent to the existence of a solution of a certain system of permutation/transformation conditions similar to those considered in [6]. Then, using the tools worked out in [6], we show that the problem of the existence of a solution of such systems is NP-hard. Since our reductions induce suitable polynomial transformations, we obtain a proof that the initial problem is co-NP-complete. In this paper, we present the first part of the plan, showing how to reduce the problem to a problem concerning systems of transformation conditions. The full proof will appear in the extended version of this paper.

2 Preliminaries

We deal with binary automata over the alphabet $\Sigma = \{\alpha, \beta\}$. The letters α and β are identified with transformations they induce on the set Q of the states. The image of $q \in Q$ by a letter (transformation) α is denoted $q\alpha$. The words $w = \alpha_1\alpha_2 \dots \alpha_t$ over Σ are identified with transformations they induce. The inverse image is denoted by $x\alpha^{-1}$.

We use special notation for concrete transformations α (similar to permutation notation), where round brackets $(x_1x_2 \dots x_t)$ denote a cycle: $x_1\alpha = x_2, \dots,$ $x_t\alpha = x_1$, and square brackets $[x_1x_2 \dots x_t]$ denote a path: $x_1\alpha = x_2, \dots, x_{t-1}\alpha = x_t$. This notation is not unique: for example $[123][42](357) = [12][423](357)$. Yet, we always write full cycles, and refer to them as the *cycles* of α, and usually omit all the fixed points. If x is an element of a cycle in α, then we write $Cyc_\alpha(x)$ to denote the cycle containing x, or a set of elements in this cycle, and we write $|Cyc_\alpha(x)|$ to denote the length of this cycle (the number of elements). We will also use a part of the structure of a transformation to speak about transformations of a given form. For example, a transformation (permutation) is of the form $\beta = (12y)(xa)(zb)\dots$ for some elements $y, x, z, a, b \in Q$ if $\beta = (12y)(xa)(zb)\tau$ for some transformation τ. We do not exclude that some of these elements may be equal, and some of these cycles coincide. For example, we may have $x = z$ and (consequently) $a = b$, in which case the cycle (xa) is the same as the cycle (zb). We may have also $x = a$, which means that (xa) is, in fact, a fixpoint $(x) = (a)$. But, assuming $1 \neq 2$, we cannot have $x = y$, because y is in a cycle of length 3, while x is in a cycle of length 2 or 1. Thus saying that a transformation is of the form $\beta = (a_1a_2 \dots a_t)\dots$ we assume that the length of the cycle is t or a divisor of t.

Treating words over $\Sigma = \{\alpha, \beta\}$ as compositions of transformations on the set Q with $1, 2 \in Q$, we shall consider systems of *transformation conditions* of the form

$$1u_1, 1u_2, \dots, 1u_s \in \{1, 2\}$$

stating that the image of 1 by each of words $u_1, u_2, \dots, u_s$ belongs to the set $\{1, 2\}$. If all transformations in Σ fix 1 or all fix the set $\{1, 2\}$, then they form a solution of the system (1), which is called *trivial.* The problem whether there exists a nontrivial solution for a system of permutation conditions has been proved to be NP-complete in [6]. Similarly one can prove that the problem is hard if we look for a solution in transformations of given types. We will exploit this in our proof.

We recall that a *factor* of a word $w \in \Sigma^+$ is a word $v \in \Sigma^+$ such that $w = uvz$ for some $u, z \in \Sigma^*$. If $\mathcal{A}$ is k-compressible at least one letter of its input alphabet has deficiency greater than 0. It is known that each k-collapsing word over a fixed alphabet Σ is *k-full* [12], that is, contains each word of length k over the alphabet Σ among its factors. A *k-compressible automaton* is called *proper* k-compressible if it cannot be k-compressed by any word of length k. Thus, k-collapsing words are k-full words that k-compress all proper k-compressible automata. In particular, in our consideration we may restrict to proper k-compressible automata.

We will consider types of transformations with regard to which states are sent into the same element, and which states are missing in the image. We say that a transformation α is of type $I \backslash M$, where M is a subset of Q, and I is a family of disjoint subsets of Q, if $M = Q \setminus Q\alpha$ is a set of elements missing in the image $Q\alpha$, while I is the family of those inverse images of elements of Q that have more than one element. In other words, we write that a letter α is of type $[x_{1_1}, \dots, x_{j_1}][x_{1_2}, \dots, x_{j_2}] \dots [x_{1_r}, \dots, x_{j_r}] \backslash y_1, y_2, ..., y_m$, if $\{y_1, \dots y_m\} = Q \setminus Im(\alpha)$ and $\{x_{1_1}, \dots, x_{j_1}\}, \{x_{1_2}, \dots, x_{j_2}\}, \dots, \{x_{1_r}, \dots, x_{j_r}\}$ are the equivalence classes of the kernel of transformation induced by α that have more than one element. We say that $\alpha \in \Sigma$ is a *permutation letter* if it induces a permutation on the set Q of the states, i.e., it has deficiency 0. Otherwise, a letter is a *non-permutation.*

The following fact leading to a classification of proper 3-compressible automata is not difficult to prove (see e.g. [1]).

Proposition 1. *If $\mathcal{A}$ is a proper 3-compressible automaton over the alphabet $\Sigma = \{\alpha, \beta\}$ then each letter in Σ is either a permutation or is one of the following types:*

1. $[x, y, z] \backslash x, y$;
2. $[x, y][z, t] \backslash x, z$;
3. $[x, y] \backslash x$;
4. $[x, y] \backslash z$ *with* $za \in \{x, y\}$.

where x, y, z, t are different states of $\mathcal{A}$.

Since we wish to classify proper 3-compressible automata up to renaming the states in Q, we may assume that $Q = \{1, 2, \dots, n\}$, and $\{x, y, z, t\} = \{1, 2, 3, 4\}$. Then, following [1], we say that an automaton $\mathcal{A}$ over a two-letter alphabet $\Sigma = \{\alpha, \beta\}$ is an $(\mathbf{i}., \mathbf{j}.)$-*automaton*, where $\mathbf{i}, \mathbf{j} \in \{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$, if the letter α is of type $\mathbf{i}.$, while the letter β is of type $\mathbf{j}.$, above. We say also that $\mathcal{A}$ is a $(\mathbf{i}., \mathbf{p})$ (or $(\mathbf{p}, \mathbf{i}.)$-*automaton*, if it is an automaton in which the letter α (β) is of type $\mathbf{i}.$, while the other letter is a permutation. By Proposition 1, up to renaming the states, each proper 3-compressible automaton over 2 letters is a $(\mathbf{t}, \mathbf{s})$-*automaton* with some $\mathbf{t}, \mathbf{s} \in \{\mathbf{1}., \mathbf{2}., \mathbf{3}., \mathbf{4}., \mathbf{p}\}$.

3 Automata of Type $(\mathbf{3.}, \mathbf{p})$

Let $\mathcal{A}$ be a $(\mathbf{3.}, \mathbf{p})$-automaton over the alphabet $\Sigma = \{\alpha, \beta\}$ and with the state set $Q = \{1, 2, \ldots, n\}$. Without loss of generality we can also assume that the letter α is of type $[1, 2]\backslash 1$ (and β is a permutation). Given such an automaton, we call an integer $k > 0$ a *good exponent* for the permutation β, or briefly, β-*good*, if $1\beta^k \notin \{1, 2\}$; otherwise, it is called β-*bad* exponent. We will consider transformation conditions of the form $1v \in \{1, 2\}$ stating the the image of 1 by the word v is 1 or 2. For a word $v \in \Sigma^+$ and $Q_1 \subseteq Q$, as in [1], we denote $\mathcal{M}(v) = Q \setminus Qv$, the set of the states missing in the image of Q under the action of v, and $\mathcal{M}(Q_1, v) = Q \setminus (Q \setminus Q_1)v$, the set of states missing in Q after applying the word v, provided that the set Q_1 of states is already missing.

Our first result is the following characterization.

Theorem 1. *Let $\mathcal{A} = \langle Q, \Sigma, \delta\rangle$ be a proper 3-compressible $(\mathbf{3.}, \mathbf{p})$-automaton with Q and Σ as above, and w be a word in Σ^+. Then, w* ***does not*** *3-compresses $\mathcal{A}$ if and only if for every factor of w of the form $\alpha u \alpha$ where*

$$u = \beta^{k_1}\alpha^{m_1}\ldots\beta^{k_t}\alpha^{m_t}\beta^{k_{t+1}},$$

$t \geq 1$, and k_1 and k_{t+1} are β-good, while all other k_i are β-bad, the condition $1u \in \{1, 2\}$ holds.

Proof. Let $w = \beta^s\alpha^{r_1}\beta^{s_1}\alpha^{r_2}\ldots\beta^{s_{h-1}}\alpha^{r_h}\beta^{s_h}$, where $s, s_h \geq 0$, $r_h > 0$, and $r_i, s_i > 0$ for $1 \leq i \leq h-1$. We consider the sequence of missed states by the action of consecutive prefixes of w. Obviously $\mathcal{M}(\beta^s\alpha^{r_1}) = \{1\}$. Moreover if s_1 is a β-bad exponent then $\mathcal{M}(\{1\}, b^{s_1}) = \mathcal{M}(\beta^s\alpha^{r_1}\beta^{s_1}) = \{1\beta^{s_1}\} \subseteq \{1, 2\}$, hence $\mathcal{M}(\beta^s\alpha^{r_1}\beta^{s_1}\alpha^{r_2}) = \{1\} = \mathcal{M}(\beta^s\alpha^{r_1})$. Repeating the same argument it turns out that if there is no β-good exponent among $s_1, s_2, \ldots, s_h$, then w does not 3-compress α.

So we may assume that there is i with $1 \leq i \leq h-1$ such that s_i is a β-good exponent and s_j is a β-bad exponent for each $j < i$. Then we have $w = p\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}}\beta^{s_{i+1}}\ldots$, where $\mathcal{M}(p\alpha^{r_i}) = \{1\}$. So $\mathcal{M}(\{1\}, \beta^{s_i}) = \{1\beta^{s_i}\} \not\subseteq \{1, 2\}$, and $\mathcal{M}(\{1\beta^{s_i}\}, \alpha^{r_{i+1}}) = \{1, x_1\}$ where $x_1 = 1\beta^{s_i}\alpha^{r_{i+1}} \neq 1$ (since α is of type $[1, 2]\backslash 1$). Now if s_{i+1} is β-bad, then $\mathcal{M}(p\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}}\beta^{s_{i+1}}) = \mathcal{M}(\{1, x_1\}, \beta^{s_{i+1}}) = \{y, x_1\beta^{s_{i+1}}\}$ with $y \in \{1, 2\}$, whence, whether $x_1\beta^{s_{i+1}} \in \{1, 2\}$ or not, we have $\mathcal{M}(p\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}}\beta^{s_{i+1}}\alpha^{r_{i+2}}) = \mathcal{M}(\{y, x_1\beta^{s_{i+1}}\}, \alpha^{r_{i+2}}) = \{1, x_2\}$ where $x_2 = x_1\beta^{s_{i+1}}\alpha^{r_{i+2}} \neq 1$. Then until we encounter β^{s_m} with a β-good exponent s_m the missing state set of each prefix of w ending with α has cardinality 2 and contains 1; in particular, if there no such second β-good exponent, w does not 3-compress α.

So, assume that there is m with $i < m \leq h$ such that s_m is a β-good exponent and each s_t with $i < t < m$ is a β-bad exponent. Then $w = p\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}}v\alpha^{r_{m+1}}\ldots$, where $v = \beta^{s_{i+1}}\alpha^{r_{i+2}}\ldots\beta^{s_m}$. Now,

$$\mathcal{M}(\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}}v) = \mathcal{M}(\{1, x_1\}, v) = \mathcal{M}(\{1, x_{m-i}\}, beta^{s_m}) = \{1\beta^{s_m}, x_{m-i}\beta^{s_m}\},$$

where $x_{m-i}\beta^{s_m} = 1v$ (since $x_1 = 1\beta^{s_i}\alpha^{r_{i+1}}, x_2 = x_1\beta^{s_{i+1}}\alpha^{r_{i+2}}$, etc.). Hence if $1v \notin \{1,2\}$ then $\mathcal{M}(\{1\beta^{s_m}, 1v\}, \alpha)$ has cardinality 3, and w is a 3-compressing word. If $1v \in \{1,2\}$ then $1v\alpha^{r_{m+1}} = 1\alpha^{r_{m+1}}$, hence $\mathcal{M}(p\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}}v\alpha^{r_{m+1}}) = \{1\beta^{s_m}, 1\alpha^{r_{m+1}}\}$. So we are in the analogous situation as with $\mathcal{M}(p\alpha^{r_i}\beta^{s_i}\alpha^{r_{i+1}})$, and we can use the above argument with i replaced by m, the prefix p replaced by $p_1 = p\alpha^{r_i}\beta^{s_i}\ldots\beta^{s_{m-1}}$, and β-good exponent s_i replaced by β-good exponent s_m. Thus, the reasoning may be repeated for consecutive factors $\alpha u \alpha$ with the property stated in the theorem, and the claim easily follows. $\square$

The theorem above means that to check whether there exists a $(\mathbf{3.}, \mathbf{p})$-automaton that fails to be 3-compressed by a given word w, we need to consider all the possibilities for β leading to various sets of good and bad exponents. This means that we consider whether 1 and 2 are in the same orbit of β. If so, we consider various lengths of this orbit and distances between 1 and 2, and if not, the only thing that counts is the length of the orbit of 1. For each such case we establish β-good and β-bad exponents, find all the factors of w of the form described in the theorem above, and form the corresponding system of transformation conditions. If the system has a solution, then w fails to 3-compress the corresponding automaton $\mathcal{A}$ with a non-permutation α and a permutation β. The solution represents an automaton that is not 3-compressed by word w. It is *nontrivial*, if the resulting automaton is 3-compressible (some solutions correspond to automata that fails to be 3-compressible).

In principle, there are infinitely many cases to consider. But given the word w, if k is the largest exponent for β (when w is written in the compact form with exponents), then only $k-1$ elements following 1 in the cycle of 1 in β are what really counts. So, in practice, for every word, we may distinguish and consider only finitely many cases.

In fact, we may view α as (almost) a permutation without 1 (1 is not an image of any state; we need only to keep in mind that 1 has the same image as 2). The system of transformation conditions may be treated very similarly to the system of suitable permutation conditions, and the method of trees with distinguished nodes developed in [6] may be used to solve it. (This is so, in spite of that the two letters involved come in the both cases from very different considerations.)

4 Reduction to a Smaller Class of Automata

To prove that solving systems of transformations in Theorem 1 is hard we need to restrict considerations to one class in which the conditions may be stated in a unique and relatively simple form. Therefore, we define the class $\mathcal{D}$ to contain all proper 3-compressible automata $\mathcal{A}$ over alphabet $\Sigma = \{\alpha, \beta\}$, such that α is a transformation of type $[1,2]\backslash 1$, and β is a permutation of the form $\beta = (12y)\ldots$. For such permutation the β-good exponents are positive integers $k = 2$ modulo 3. We will refer to them as to $(12y)$-good. Other positive integers are $(12y)$-bad.

In order to reduce our problem to automata in $\mathcal{D}$, to each word $w \in \{\alpha, \beta\}^+$ we assign a word $w^* = w'w$ such that w^* is 3-collapsing if and only if w 3-compresses all automata in $\mathcal{D}$. The idea is to find a set of words that 3-compress all automata not in $\mathcal{D}$, and do not 3-compress automata in $\mathcal{D}$. These words will be used to form w'. We wish that these words have no $(12y)$-good exponents, because then w^* and w have the same factors described in Theorem 1, which guarantees that they 3-compress the same proper automata in $\mathcal{D}$. This cannot be achieved exactly, and the few exceptions we allow will be handled further in a different way.

We start from $(\mathbf{3.}, \mathbf{p})$-automata not in $\mathcal{D}$. The following technical lemma established in [1] will be useful here.

Lemma 1. ([1], Lemma 3) *Let $\mathcal{A}$ be a $(\mathbf{3.}, \mathbf{p})$-automaton with α of type* $[1,2]\backslash 1$. *Then $\mathcal{A}$ is not 3-compressible if, and only if (up to renaming the states) one of the following conditions holds:*

(i) *β fixes 1 or the set $\{1,2\}$,*
(ii) *$\beta = (13)\pi$ for some permutation π on $Q \setminus \{1,3\}$ and $3\alpha = 3$,*
(iii) *$\beta = (13)(2)\pi$, $\beta = (132)\pi$ or $\beta = (123)\pi$ for some permutation π on $Q \setminus \{1,2,3\}$ and $\{2,3\}\alpha = \{2,3\}$,*
(iv) *$\beta = (13)(24)\pi$ or $\beta = (1324)\pi$ for some permutation π on $Q \setminus \{1,2,3,4\}$ and $\{3,4\}\alpha = \{3,4\}$.*

In all other cases $\mathcal{A}$ is a proper 3-compressible automaton, and one of the words $\{ababa$, $abab^2a$, aba^2ba, $abab^2aba$, ab^2ab^2a, $ab^2a^2b^2a$, ab^2abab^2a, ab^2aba, ab^3aba, $abab^3a$, $ab^3ab^3a\}$ 3-compresses $\mathcal{A}$.

Using it we get the following.

Lemma 2. *Each proper 3-compressible $(\mathbf{3.}, \mathbf{p})$-automaton $\mathcal{A} \notin \mathcal{D}$ can be 3-compressed by a word of the form $\alpha\beta^i\alpha^k\beta^j\alpha$, where $i,j \in \{1,3,4\}, k \in \{1,2,3\}$ or $\alpha\beta^4\alpha\beta^2\alpha\beta^4\alpha$.*

Proof. Let $\mathcal{A}$ be a $(\mathbf{3.}, \mathbf{p})$-automaton $\mathcal{A} \notin \mathcal{D}$, then by (i) of Lemma 1, b fixes neither 1 nor the set $\{1,2\}$.

First, assume that $1\beta = x \notin \{1,2\}$. Then, $\mathcal{M}(\alpha\beta\alpha^k) = \{1, x\alpha^k\}$. If $x\alpha^k\beta \notin \{1,2\}$ for some k then the word $\alpha\beta\alpha^k\beta\alpha$ 3-compresses the automaton $\mathcal{A}$. Obviously, if $x\alpha^k\beta \notin \{1,2\}$, for some k we can always assume that $k \leq 3$. Otherwise, either $x\alpha = x$ or $x\alpha^2 = x$ and $x\alpha^k\beta \in \{1,2\}$ for all k.

Then let $x\alpha^k\beta \in \{1,2\}$ for each $k \in \{1,2,3\}$.

First assume $x\alpha = x$. If $x\alpha\beta = x\beta = 1$ then $\beta = (1x)\ldots$ and by (ii), $\mathcal{A}$ is not 3-compressible. So $x\alpha\beta = x\beta = 2$ and $\beta = (1x2\ldots)\ldots$. If $\beta = (1x2)\ldots$ then $2\alpha \neq 2$ otherwise $\mathcal{A}$ is not 3-compressible by (iii). Then $\mathcal{M}(\alpha\beta^{1+3h}\alpha\beta^2) = \{1,2\}$ for each $h \geq 0$. Since $\mathcal{M}(\{1,2\},\alpha) = \{1,2\alpha\}$ and $2\alpha \notin \{1,2,x\}$, then $\alpha\beta^{1+3h}\alpha\beta^2\alpha\beta^{1+3k}\alpha$ 3-compresses $\mathcal{A}$. So let $\beta = (1x2y\ldots)\ldots$. Then $\mathcal{M}(\alpha\beta^3\alpha) = \{1, y\alpha\}$ and if $y\alpha\beta \notin \{1,2\}$ then $\alpha\beta^3\alpha\beta\alpha$ 3-compresses $\mathcal{A}$. We know that $y\alpha\beta \neq x\alpha\beta = 2$, so let $y\alpha\beta = 1$. If $y\alpha = y$ then $\mathcal{A}$ is not 3-compressible by (iv). If $y\alpha \neq y$ then $\alpha\beta^3\alpha\beta^4\alpha$ 3-compresses $\mathcal{A}$.

Then let $x\alpha \neq x$ and $x\alpha^2 = x$. Put $x\alpha = y$, then $y\alpha = x$ and $y\alpha\beta = x\alpha^2\beta = x\beta \in \{1,2\}$. If $x\beta = 1$ then $\beta = (1x)(y2z\ldots)\ldots$ where y, z, 2 are all distinct by (iii) and (iv). Hence $\alpha\beta\alpha\beta^3\alpha$ 3-compresses $\mathcal{A}$. So let $x\beta \neq 1$, hence $x\beta = 2$, and again by (iv), $\beta = (1x2z\dots y)\dots$ where $1, x, 2, z, y$ are distinct states, then $\alpha\beta\alpha\beta^4\alpha$ is a 3-compressing $\mathcal{A}$. This completes all the cases when $1\beta = x \neq 1, 2$.

Lastly, let $1\beta = 2$. Then $\beta = (12xy\dots)\dots$ where $1, 2, x, y$ are distinct elements, otherwise $\mathcal{A} \in \mathcal{D}$. Then $\mathcal{M}(\alpha\beta^3\alpha) = \{1, y\alpha\}$. If $y\alpha\beta^3 \notin \{1, 2\}$ then $\alpha\beta^3\alpha\beta^3\alpha$ is 3-compressing, similarly if $y\alpha\beta^4 \notin \{1, 2\}$ then $\alpha\beta^3\alpha\beta^4\alpha$ is 3-compressing. So $y\alpha\beta^3 = 1$ and $y\alpha\beta^4 = 2$ whence $y \neq y\alpha$ and so $y\alpha \neq y\alpha^2$, and so $y\alpha^2\beta^3 \neq 1$. If $y\alpha^3\beta^3 \neq 2$ then $\alpha\beta^3\alpha^2\beta^3\alpha$ is 3-compressing, if $y\alpha^3\beta^3 = 2$, then $y\alpha^3\beta^4 = x$ and $\alpha\beta^3\alpha^2\beta^4\alpha$ is 3-compressing. □

For the automata of types other than $(\mathbf{3.}, \mathbf{p})$ we make use of Proposition 1 and other lemmas established in [1]. These lemmas have been established in [1] to compute a bound for the length of the shortest 3-collapsing word. We extract from them information we need to our aim. We have the following.

(i) no 3-compressible $(\mathbf{i.}, \mathbf{j.})$-automaton with $\mathbf{i} \in \{\mathbf{1}, \mathbf{2}\}$, $\mathbf{j} \in \{\mathbf{1}, \mathbf{2}, \mathbf{4}\}$ or with $\mathbf{j} \in \{\mathbf{1}, \mathbf{2}\}$, $\mathbf{i} \in \{\mathbf{1}, \mathbf{2}, \mathbf{4}\}$ is proper; [1, Lemma 5];

(ii) each proper 3-compressible $(\mathbf{1.}, \mathbf{p})$-automaton and each proper 3-compressible $(\mathbf{1.}, \mathbf{3})$-automaton is 3-compressed by $\underline{\alpha\beta^2\alpha}$; hence (by switching the letters), each proper 3-compressible $(\mathbf{p.}, \mathbf{1})$ or $(\mathbf{3.}, \mathbf{1})$-automaton is 3 compressed by $\beta\alpha^2\beta$; [1, Lemma 1 and Lemma 6];

(iii) each proper 3-compressible $(\mathbf{2.}, \mathbf{p})$ and each proper 3-compressible $(\mathbf{2.}, \mathbf{3})$-automaton is 3-compressed by a word in the set $\{\underline{\alpha\beta^2\alpha}, \alpha\beta^3\alpha\}$; each proper 3-compressible $(\mathbf{p.}, \mathbf{2})$ and each proper 3-compressible $(\mathbf{3.}, \mathbf{2})$-automaton is 3-compressed by a word in the set $\{\beta\alpha^2\beta, \beta\alpha^3\beta\}$; [1, Lemma 2 and Lemma 7];

For the future reference we underline words that have an occurrence of β with a $(12y)$-good exponent *inside* the word. Here, this is limited only to occurrences of β^2. These words will be handled in a different way than those words that have no occurrence β with a $(12y)$-good exponent or they have such an occurrence only at the beginning or at the end of the word. We will see that only the exact form of underlined words is what really counts in our proof.

In order to find a set of words 3-compressing all $(\mathbf{p}, \mathbf{3.})$-automata we use again Lemma 1, yet switching the letters α and β. It yields the following set: $\{\beta\alpha\beta\alpha\beta, \beta\alpha\beta\alpha^2\beta, \beta\alpha\beta^2\alpha\beta, \beta\alpha\beta\alpha^2\beta\alpha\beta, \beta\alpha^2\beta\alpha^2\beta, \beta\alpha^2\beta^2\alpha^2\beta, \beta\alpha^2\beta\alpha\beta\alpha^2\beta, \beta\alpha^2\beta\alpha\beta, \beta\alpha^3\beta\alpha\beta, \beta\alpha\beta\alpha^3\beta, \beta\alpha^3\beta\alpha^3\beta\}$. Some of these words are factors of others, so we may infer the following:

(iv) each proper 3-compressible $(\mathbf{p}, \mathbf{3.})$-automaton is 3-compressed by any word with a factor of the form $\beta\alpha\beta\alpha\beta, \underline{\beta\alpha\beta^2\alpha\beta}, \beta\alpha\beta\alpha^2\beta\alpha\beta, \beta\alpha^2\beta\alpha^2\beta, \underline{\beta\alpha^2\beta^2\alpha^2\beta}$, $\beta\alpha^2\beta\alpha\beta\alpha^2\beta, \beta\alpha^3\beta\alpha\beta, \beta\alpha\beta\alpha^3\beta, \beta\alpha^3\beta\alpha^3\beta$; [1, Lemma 3];

Similarly we get the following

(v) each proper 3-compressible $(\mathbf{4.}, \mathbf{p})$ is 3-compressed by any word with a factor of the form $\alpha\beta\alpha\beta\alpha, \underline{\alpha^2\beta^2\alpha^2}, \alpha^2\beta\alpha^2, \alpha^2\beta^3\alpha, \alpha\beta^3\alpha\beta^3\alpha, \underline{\alpha^2\beta\alpha\beta^2\alpha}$; each proper 3-compressible $(\mathbf{p.}, \mathbf{4})$ is 3-compressed by any word with a factor of the form $\beta\alpha\beta\alpha\beta, \beta^2\alpha^2\beta^2, \beta^2\alpha\beta^2, \beta^2\alpha^3\beta, \beta\alpha^3\beta\alpha^3\beta, \beta^2\alpha\beta\alpha^2\beta$; [1, Lemma 4];

(vi) each proper 3-compressible $(\mathbf{3.},\mathbf{3})$-automaton is 3-compressed by a word in the set $\{\alpha\beta\alpha\beta, \underline{\alpha\beta^2\alpha\beta}, \alpha\beta\alpha^2\beta, \underline{\alpha\beta^2\alpha^2\beta}, \beta\alpha\beta\alpha, \beta\alpha^2\beta\alpha, \underline{\beta\alpha\beta^2\alpha}, \underline{\beta\alpha^2\beta^2\alpha}\}$; [1, Lemma 9];

(vii) each proper 3-compressible $(\mathbf{3.},\mathbf{4})$-automaton is 3-compressed by a word in the set $\{\beta^2\alpha\beta^2, \beta^2\alpha^2\beta^2, \beta^2\alpha^3\beta^2, \beta^2\alpha\beta\alpha\beta^2\}$; each proper 3-compressible $(\mathbf{4.},\mathbf{3})$-automaton is 3-compressed by a word in the set $\{\alpha^2\beta\alpha^2, \underline{\alpha^2\beta^2\alpha^2}, \alpha^2\beta^3\alpha^2, \alpha^2\beta\alpha\beta\alpha^2\}$; [1, Lemma 10];

(viii) each proper 3-compressible $(\mathbf{4.},\mathbf{4})$ is 3-compressed by a word in the set $\{\alpha^2\beta\alpha^2, \alpha^2\beta^2, \beta^2\alpha\beta^2, \beta^2\alpha^2\}$; [1, Lemma 11].

Using the lemmas above we can see that any word containing as factors the following words

(I) $\underline{\alpha^2\beta\alpha\beta^2\alpha}, \underline{\beta\alpha^2\beta^2\alpha^2\beta}, \underline{\alpha\beta^4\alpha\beta^2\alpha\beta^4\alpha},$

(II) $\alpha\beta^3\alpha\beta^3\alpha, \beta\alpha^3\beta\alpha^3\beta, \beta^2\alpha\beta\alpha^2\beta, \beta\alpha\beta\alpha^2\beta\alpha\beta, \beta\alpha^2\beta\alpha^2\beta, \beta\alpha^2\beta\alpha\beta\alpha^2\beta, \beta\alpha^3\beta\alpha\beta, \beta\alpha\beta\alpha^3\beta, \beta\alpha^3\beta\alpha^3\beta, \beta^2\alpha^2\beta^2, \beta^2\alpha^3\beta^2, \beta^2\alpha\beta\alpha\beta^2, \alpha^2\beta^3\alpha^2.$

(III) $\alpha\beta^i\alpha^k\beta^j\alpha$, where $i,j \in \{1,3,4\}, k \in \{1,2,3\}$.

3-compresses all 3-compressible automata except those in the $\mathcal{D}$.

To form a single word that 3-compresses all automata not in $\mathcal{D}$ it is enough to concatenate all the words listed in (I-III) above. Yet, we wish to have such a word without $(12y)$-good exponents. So, at this moment, we have only a partial solution. Let $w_{\mathcal{D}}$ be the word obtained from concatenation of words in (II) and (III), in arbitrary order, adding the suffix $\alpha^2\beta\alpha$ at the beginning, and replacing all β^2 by β^3 (more precisely replacing all factors $\alpha\beta^2\alpha$ by $\alpha\beta^3\alpha$; note that the factors $\alpha\beta^2\alpha$ in the word obtained from concatenation of words in (II) and (III) can only come from the concatenation of words in (II-III) ending and starting with β). Then obviously, $w_{\mathcal{D}}$ has all words in (II-III) as factors and has no occurrences of $(12y)$-good exponents. The fact that it has the suffix $\alpha^2\beta\alpha$ will be used later in the proof. We observe that $w_{\mathcal{D}}$ is also 3-full, since all words of length 3 appear as factors in words listed in (II). This means we may state the following.

Proposition 2. *Let $w \in \{\alpha,\beta\}^*$ be a word such that $w^* = w_{\mathcal{D}}w$ has as factors all the three words listed in (I) above. Then, w 3-compresses all proper automata in $\mathcal{D}$ if and only if the word $w^* = w_{\mathcal{D}}w$ described above is 3-collapsing.*

5 Reduction to a System of Transformation Conditions

Now, our aim is to express the problem of 3-compressibility of proper automata in $\mathcal{D}$ in terms of solving a system of transformation conditions. The base for this is Theorem 1. Our attention is restricted to transformations α and β satisfying the following conditions.

(C1) α is a transformation of type $[1,2]\backslash 1$;

(C2) β is a permutation of the form $\beta = (12y)\dots$ for some $y \notin \{1,2\}$;

(C3) either 2α or $y\alpha$ is not in $\{2,y\}$.

Given a word $w \in \{\alpha, \beta\}^+$, by $u_1, u_2, \dots, u_s$ we denote the set of all factors of w such that $\alpha u_1 \alpha$, $\alpha u_2 \alpha$, $\dots$, $\alpha u_s \alpha$ are all the factors of w defined in Theorem 1 for the permutation $\beta = (12y)\dots$. Then we have the following.

Proposition 3. *Let $w \in \{\alpha, \beta\}^+$, and let $u_1, u_2, \dots, u_s$ be the factors of w described above. Then, there exists a proper automaton $A \in \mathcal{D}$ such that w does not 3-compresses A if and only if the system*

$$1u_1, 1u_2, \dots, 1u_s \in \{1, 2\} \tag{1}$$

has a solution in transformations α, β on a finite set $Q = \{1, 2, \dots, n\}$ satisfying the conditions (C1-C3).

Proof. First, suppose that a required solution exists, and let $\mathcal{A}$ be an automaton with the state set $Q = \{1, 2, \dots, n\}$ and two input letters whose transition function is defined by the action of the letters α and β given by the solution. Then conditions (C1) and (C2) mean simply that $A \in \mathcal{D}$, provided it is proper 3-compressible. It is 3-compressible by Lemma 1, item (iii). Finally, it is not difficult to see that any word 3-compressing A has a length exceeding 4. Indeed, we need first a letter α to get 1 missing in the image, then a factor β^2, to get y missing in the image, and the next α, to get two states missing. Thus $A \in \mathcal{D}$ and, by Theorem 1, w does not 3-compress $\mathcal{A}$.

Conversely, if $A \in \mathcal{D}$ and w does not 3-compresses it, then we may assume that its set of the states is $Q = \{1, 2, \dots, n\}$. Then, the transformations α and β corresponding to the letters of $\mathcal{A}$ satisfy, by definition of $\mathcal{D}$, the condition (C1) and (C2), and as above, by Lemma 1, they satisfy also condition (C3). Thus, by Theorem 1, they form a required solution of the system (1). □

In such a way the problem of 3-compressibility of automata in $\mathcal{D}$ is reduced to solving a system of transformation conditions $1u \in \{1, 2\}$ with all u of the form $u = \beta^k \alpha u' \alpha \beta^\ell$, where k, ℓ are $(12y)$-good exponents, and u' has no occurrence of $(12y)$-bad exponents. We are going to show that solving a certain subclass of such systems is computationally hard.

Given words $v_1, \dots, v_s \in \{\alpha, \beta\}^+$, we add to them two further words $v_0 = \alpha^2$ and $v_{s+1} = \alpha\beta^4\alpha$, and consider the following system of transformation conditions.

$$1\beta^2 v_0 \beta^2,\ 1\beta^2 v_1 \beta^2,\ \dots,\ 1\beta^2 v_{s+1} \beta^2 \in \{1, 2\} \tag{2}$$

We define a specific decision problem:

PROBLEM (*)
Instance: words $v_1, \dots, v_s \in \{\alpha, \beta\}^+$ such that each word v_i starts and ends with α, and has no occurrence of β with a $(12y)$-good exponent;
Question: Is there a solution (α, β) of the system (2) satisfying conditions (C1-C3)?

We have the following.

Theorem 2. *Problem (*) formulated above is NP-complete.*

The proof ot this theorem uses tools worked out in [6], where solving systems of such conditions is expressed in terms of coloring trees with distinguished nodes. It will be given in the extended version of the paper. Having this theorem we can can easily prove our main result.

Theorem 3. *The problem whether a given word $w \in \{\alpha, \beta\}^*$ is 3-collapsing is co-NP-complete.*

Proof. First observe that the problem belongs to co-NP class. Indeed, to see that a word w is not 3-collapsing a nondeterministic algorithm needs only to guess the smallest automaton that is not 3-compressed by w. By [10, Theorem 1] such an automaton has not more than $4|w| + 2$ states, and the facts that it is 3-compressible and that w does not 3-compress it can be checked easily in polynomial time with respect to $|w|$.

We transform problem (*) to our problem. Let $v_1, \ldots, v_s$ be an instance of (*). First we form the word

$$w' = \beta^2 v_0 \beta^2 v_0 \beta^2 v_1 \beta^2 v_2 \beta^2 \ldots \beta^2 v_s \beta^2 v_{s+1} \beta^2 v_{s+1} \beta^2.$$

Note that this word has doubled occurrences of factors v_0 and v_{s+1}. By assumption, the only occurrences of β in w' with $(12y)$-good exponents are β^2 separating factors $v_0, v_1, v_2, \ldots, v_s, v_{s+1}$. Thus, by Proposition 3, system (2) has a solution satisfying conditions (C1-C3) if and only if there exists a proper automaton $A \in \mathcal{D}$ such that w does not 3-compress A.

Now, we observe that $w = w_{\mathcal{D}} w'$, defined as in Proposition 2, has as factors all the three words listed in (I). Indeed, $\alpha^2\beta\alpha\beta^2\alpha$ is a factor of w since $w_{\mathcal{D}}$, by definition, has the suffix $\alpha^2\beta\alpha$, and w' starts from $\beta^2\alpha$. The prefix $\beta^2 v_0 \beta^2 v_0 \beta^2 = \beta^2\alpha^2\beta^2\alpha^2\beta^2$ of w' has the second word in (I) as a factor. Finally, the suffix $v_{s+1}\beta^2 v_{s+1}\beta^2 = \alpha\beta^4\alpha\beta^2\alpha\beta^4\alpha\beta^2$ of w' has the third word in (I) as a factor. Therefore, by Proposition 2, there exists a proper automaton $A \in \mathcal{D}$ such that w does not 3-compresses A if and only if the word $w^* = w_{\mathcal{D}} w$ is not 3-collapsing. Obviously, this transformation may be performed in polynomial time, which completes the proof. □

References

1. A. Frigeri, A. Cherubini and Z. Liu. Composing short 3-compressing words on a 2 letter alphabet. *to appear,* (arxiv.org: 1406.1413v1, 2014).
2. D. S. Ananichev, A. Cherubini, and M. V. Volkov. Image reducing words and subgroups of free groups. *Theor. Comput. Sci.*, 307(1):77–92, 2003.
3. D. S. Ananichev, A. Cherubini, and M. V. Volkov. An inverse automata algorithm for recognizing 2-collapsing words. In *Developments in Language Theory*, volume 2450 of *LNCS*, pages 270–282, 2003.
4. D. S. Ananichev and I. V. Petrov. Quest for short synchronizing words and short collapsing words. In *WORDS. Proc. 4th Int. Conf.*, pages 411–418, 2003.

5. A. Cherubini, P. Gawrychowski, A. Kisielewicz, and B. Piochi. A combinatorial approach to collapsing words. In *MFCS*, pages 256–266, 2006.
6. A. Cherubini and A. Kisielewicz. Collapsing words, permutation conditions and coherent colorings of trees. *Theor. Comput. Sci.*, 410(21-23):2135–2147, 2009.
7. J. E. Pin. Le problème de la synchronization. contribution à l'étudia de la conjecture de Černý. *Thèse 3e cycle, Paris*, 1978.
8. J. E. Pin. Sur le mots synchronisants dans un automata fini. *Elektron. Informationverarbeigtung und Kybernetik*, 14:283–289, 1978.
9. S. W. Margolis, J.-E. Pin, and M. V. Volkov. Words guaranteeing minimum image. *Internat. J. Foundations Comp. Sci.*, 15:259–276, 2004.
10. I. V. Petrov. An algorithm for recognition of n-collapsing words. *Theoret. Comput. Sci.*, 391(1-2):99–108, 2008.
11. E. V. Pribavkina. On some properties of the language of 2-collapsing words. In *Developments in Language Theory*, volume 3572 of *LNCS*, pages 374–384, 2005.
12. N. Sauer and M. G. Stone. Composing functions to reduce image size. *Ars Combinatoria*, 1:171–176, 1991.

Extendibility of Choquet rational preferences on generalized lotteries

Giulianella Coletti[1], Davide Petturiti[2], and Barbara Vantaggi[2]

[1] Dip. Matematica e Informatica, Università di Perugia, Italy
coletti@dmi.unipg.it
[2] Dip. S.B.A.I., Università di Roma "La Sapienza", Italy
{davide.petturiti,barbara.vantaggi}@sbai.uniroma1.it

Abstract. Given a finite set of generalized lotteries, that is random quantities equipped with a belief function, and a partial preference relation on them, a necessary and sufficient condition (Choquet rationality) has been provided for its representation as a Choquet expected utility of a strictly increasing utility function. Here we prove that this condition assures the extension of the preference relation and it actually guides the decision maker in this process.

Keywords: Generalized lottery, preference relation, belief function, probability envelope, Choquet expected utility, Choquet rationality

1 Introduction

In the classical von Neumann-Morgenstern decision theory under risk [23, 18], the decision maker faces "one-shot" decisions [17] by specifying a preference relation on *lotteries*, i.e., random quantities endowed with a probability distribution. If the preference relation satisfies suitable axioms then the preference is representable by an *expected utility* (EU) and the decision maker behaves like an EU maximizer.

The assumptions behind the EU theory rely on a complete probabilistic description of the decisions, which is rarely met in practice. Indeed, in situations of incomplete and revisable information, uncertainty cannot be handled through a probability but it is unavoidable to refer to non-additive uncertainty measures, for which the EU model is no more appropriate.

Here, we refer to Dempster-Shafer *belief functions* [7, 19] as uncertainty measures and to *Choquet expected utility* (CEU) as decision model (see for instance [20, 21, 15, 1]). We recall that in some probabilistic inferential problems belief functions can be obtained as lower envelopes of a family of probabilities, possibly arising as coherent extensions of a probability assessed on a set of events different from those of interest (see for instance [7, 5, 10, 14, 6]).

Another issue typical of real problems is the partial observability of the world which leads the decision maker to act under partial knowledge. Both in the classical expected utility and in the Choquet expected utility frameworks it can be difficult to construct the utility function u and even to test if the preferences agree with an EU (or a CEU). In fact, to find the utility u the classical methods ask for comparisons between "lotteries" and "certainty equivalent" or, in any case, comparisons among particular large classes of lotteries (for a discussion in the EU framework see [13]). For that, the decision maker is often forced to make comparisons which have little or nothing to do with the given problem, having to choose between risky prospects and certainty.

In [4], referring to the EU model, a different approach (based on a "rationality principle") is proposed: it does not need all these non-natural comparisons but, instead, it can work by considering only the (few) lotteries and comparisons of interest. Moreover, when new information is introduced, the same principle assures that the preference relation can be extended maintaining rationality, and, even more, the principle suggests how to extend it.

In [2] and in an extended version [3], we proposed a similar approach for the CEU model by generalizing the usual definition of lottery. In detail, a *generalized lottery* L (or *g-lottery* for short) is a random quantity with a finite support X_L endowed with a Dempster-Shafer belief function Bel_L [7, 19, 22] (or, equivalently, a *basic assignment* m_L) defined on the power set $\wp(X_L)$.

Assuming that the elements of the set $X = \{x_1, \ldots, x_n\}$ resulting by the union of the supports of the considered g-lotteries is totally ordered as $x_1 < \ldots < x_n$ (which is quite natural, thinking at elements of X as money payoffs), then for every g-lottery L the Choquet integral of any strictly increasing utility function $u : X \to \mathbb{R}$, not only is a weighted average (as observed in [12]), but the weights have a clear meaning. In fact, this allows to map every g-lottery L to a "standard" lottery whose probability distribution is constructed (following a pessimistic approach) through the *aggregated basic assignment* M_L.

The "Choquet rationality principle" (namely, condition **(g-CR)**) requires that *it is not possible to obtain two g-lotteries L and L' with $M_L = M_{L'}$, by combining in the same way the aggregated basic assignments of two groups of g-lotteries, if every g-lottery of the first group is not preferred to the corresponding one of the second group, and at least a preference is strict.*

Condition **(g-CR)** turns out to be necessary and sufficient for the existence of a strictly increasing $u : X \to \mathbb{R}$ whose CEU represents our preferences on a finite set $\mathcal{L}$ of g-lotteries, under a natural assumption of agreement of the preference relation with the order of X.

In this paper we show that condition **(g-CR)** assures also the extendibility of a preference relation and actually "guides" the decision maker in this process. An algorithm for the extension of a preference relation to a new pair of g-lotteries is also provided. Such algorithm relies on the solution of at most three linear programming problems and can be used "interactively" by the decision maker in a step by step enlargement of his preferences.

The paper is structured as follows. In Section 2 some preliminary notions are given, while Section 3 copes with preferences on g-lotteries and introduces the condition **(g-CR)**. Finally, Subsection 3.1 presents a motivating example, and Subsection 3.2 deals with the extendibility of a Choquet rational preference relation providing an algorithm for this task.

2 Numerical model of reference

Let X be a finite set of states of nature and denote by $\wp(X)$ the power set of X. We recall that a *belief function Bel* [7, 19, 22] on an algebra of events $\mathcal{A} \subseteq \wp(X)$ is a function such that $Bel(\emptyset) = 0$, $Bel(X) = 1$ and satisfying the n-monotonicity property for every $n \geq 2$, i.e., for every $A_1, \ldots, A_n \in \mathcal{A}$,

$$Bel\left(\bigcup_{i=1}^{n} A_i\right) \geq \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} Bel\left(\bigcap_{i \in I} A_i\right). \tag{1}$$

A belief function Bel on $\mathcal{A}$ is completely singled out by its Möbius inverse, defined for every $A \in \mathcal{A}$ as

$$m(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|} Bel(B).$$

Such a function, usually called *basic (probability) assignment*, is a function $m : \mathcal{A} \to [0,1]$ satisfying $m(\emptyset) = 0$ and $\sum_{A \in \mathcal{A}} m(A) = 1$, and is such that for every $A \in \mathcal{A}$

$$Bel(A) = \sum_{B \subseteq A} m(B). \tag{2}$$

A set A in $\mathcal{A}$ is a *focal element* for m (and so also for the corresponding Bel) whenever $m(A) > 0$.

Given a set $X = \{x_1, \ldots, x_n\}$ and a normalized capacity $\varphi : \wp(X) \to [0,1]$ (i.e., a function monotone with respect to the inclusion, and satisfying $\varphi(\emptyset) = 0$ and $\varphi(X) = 1$), the *Choquet integral* of a function $f : X \to \mathbb{R}$, with $f(x_1) \leq \ldots \leq f(x_n)$ is defined as

$$\oint f \, \mathrm{d}\varphi = \sum_{i=1}^{n} f(x_i)(\varphi(E_i) - \varphi(E_{i+1})) \tag{3}$$

where $E_i = \{x_i, \ldots, x_n\}$ for $i = 1, \ldots, n$, and $E_{n+1} = \emptyset$ [8].

In the classical von Neumann-Morgenstern theory [23] a *lottery* L consists of a *probability distribution* on a finite *support* X_L, which is an arbitrary finite set of *prizes* or *consequences*.

In this paper we adopt a generalized notion of lottery L, by assuming that a *belief function* Bel_L is assigned on the power set $\wp(X_L)$ of X_L.

Definition 1. *A* **generalized lottery**, *or* **g-lottery** *for short, on a finite set* X_L *is a pair* $L = (\wp(X_L), Bel_L)$ *where* Bel_L *is a belief function on* $\wp(X_L)$.

Let us notice that, a g-lottery $L = (\wp(X_L), Bel_L)$ could be equivalently defined as $L = (\wp(X_L), m_L)$, where m_L is the basic assignment associated to Bel_L. We stress that this definition of g-lottery generalizes the classical one in which $m_L(A) = 0$ for every $A \in \wp(X_L)$ with $\operatorname{card} A > 1$.

For example, a g-lottery L on $X_L = \{x_1, x_2, x_3\}$ can be expressed as

$$L = \begin{pmatrix} \{x_1\} & \{x_2\} & \{x_3\} & \{x_1, x_2\} & \{x_1, x_3\} & \{x_2, x_3\} & \{x_1, x_2, x_3\} \\ b_1 & b_2 & b_3 & b_{12} & b_{13} & b_{23} & b_{123} \end{pmatrix}$$

where the belief function Bel_L on $\wp(X_L)$ is such that $b_I = Bel_L(\{x_i : i \in I\})$ for every $I \subseteq \{1, 2, 3\}$. Notice that as one always has $Bel_L(\emptyset) = m_L(\emptyset) = 0$, the empty set is not reported in the tabular expression of L. An equivalent representation of previous g-lottery is obtained through the basic assignment m_L associated to Bel_L (where $m_I = m_L(\{x_i : i \in I\})$ for every $I \subseteq \{1, 2, 3\}$)

$$L = \begin{pmatrix} \{x_1\} & \{x_2\} & \{x_3\} & \{x_1, x_2\} & \{x_1, x_3\} & \{x_2, x_3\} & \{x_1, x_2, x_3\} \\ m_1 & m_2 & m_3 & m_{12} & m_{13} & m_{23} & m_{123} \end{pmatrix}.$$

Given a finite set $\mathcal{L}$ of g-lotteries, let $X = \bigcup\{X_L : L \in \mathcal{L}\}$. Then, any g-lottery L on X_L with belief function Bel_L can be rewritten as a g-lottery on X by defining a suitable extension Bel'_L of Bel_L.

Proposition 1. *Let $L = (\wp(X_L), Bel_L)$ be a g-lottery on X_L. Then for any finite $X \supseteq X_L$ there exists a unique belief function Bel'_L on $\wp(X)$ with the same focal elements of Bel_L and such that $Bel'_{L|\wp(X_L)} = Bel_L$.*

Given $L_1, \ldots, L_t \in \mathcal{L}$, all rewritten on X, and a real vector $\mathbf{k} = (k_1, \ldots, k_t)$ with $k_i \geq 0$ $(i = 1, \ldots, t)$ and $\sum_{i=1}^t k_i = 1$, the *convex combination* of $L_1, \ldots, L_t$ according to $\mathbf{k}$ is defined as

$$\mathbf{k}(L_1, \ldots, L_t) = \begin{pmatrix} A \\ \sum_{i=1}^t k_i m_{L_i}(A) \end{pmatrix} \quad \text{for every } A \in \wp(X) \setminus \{\emptyset\}. \tag{4}$$

Since the convex combination of belief functions (basic assignments) on $\wp(X)$ is a belief function (basic assignment) on $\wp(X)$, $\mathbf{k}(L_1, \ldots, L_t)$ is a g-lottery on X.

For every $A \in \wp(X) \setminus \{\emptyset\}$, there exists a *degenerate g-lottery* δ_A on X such that $m_{\delta_A}(A) = 1$, and, moreover, every g-lottery L with focal elements $A_1, \ldots, A_k$ can be expressed as $\mathbf{k}(\delta_{A_1}, \ldots, \delta_{A_k})$ with $\mathbf{k} = (m_L(A_1), \ldots, m_L(A_k))$.

3 Preferences over a set of generalized lotteries

Consider a set $\mathcal{L}$ of g-lotteries with $X = \bigcup\{X_L : L \in \mathcal{L}\}$ and assume X is *totally ordered* by the relation $\leq$, which is a quite natural condition thinking at elements of X as money payoffs. Denote with $<$ the total strict order on X induced by $\leq$.

In what follows the set X is always assumed to be finite, i.e., $X = \{x_1, \ldots, x_n\}$ with $x_1 < \ldots < x_n$. Under previous assumption, we can define the *aggregated basic assignment* of a g-lottery L, for every $x_i \in X$, as

$$M_L(x_i) = \sum_{x_i \in B \subseteq E_i} m_L(B), \tag{5}$$

where $E_i = \{x_i, \ldots, x_n\}$ for $i = 1, \ldots, n$. Note that $M_L(x_i) \geq 0$ for every $x_i \in X$ and $\sum_{i=1}^{n} M_L(x_i) = 1$, thus M_L determines a probability distribution on X.

Let $\precsim$ be a *preference/indifference* relation on $\mathcal{L}$. For every $L, L' \in \mathcal{L}$ the assertion that "L is indifferent to L'", denoted by $L \sim L'$, summarizes the two assertions $L \precsim L'$ and $L' \precsim L$. Observe that not all the pairs of g-lotteries are necessarily compared. An additional strict preference relation can be elicited by assertions such as "L is strictly preferred to L'", denoted by $L \prec L'$. Let $\prec^\bullet$ be the asymmetric relation formally deduced from $\precsim$, namely $\prec^\bullet = \precsim \setminus \sim$. If the pair of relations $(\precsim, \prec)$ represents the opinion of the decision maker, then it is natural to have $\prec \subset \prec^\bullet$: in fact, it is possible that, at an initial stage of judgement, the decision maker has not decided yet if $L \prec L'$ or $L \sim L'$ and he expresses his opinion only by $L \precsim L'$. Obviously if $\precsim$ is complete then $\prec = \prec^\bullet$ and so for every $L, L' \in \mathcal{L}$ either $L \prec L'$ or $L' \prec L$ or $L \sim L'$.

Remark 1. Since the set X is totally ordered by $\leq$, it is natural to require that the partial preference relation $(\precsim, \prec)$ agrees with $\leq$ on degenerate g-lotteries $\delta_{\{x\}}$, for $x \in X$, that correspond to decisions under certainty. For this, $\mathcal{L}$ must contain the set of degenerate g-lotteries on singletons $\mathcal{L}_0 = \{\delta_{\{x\}} : x \in X\}$ and it must be $x \leq x'$ if and only if $\delta_{\{x\}} \precsim \delta_{\{x'\}}$, for $x, x' \in X$. Actually, the decision maker is not asked to provide such a set of preferences, but in this case the initial partial preference $(\precsim, \prec)$ on $\mathcal{L}$ must be extended in order to reach this technical condition and, of course, the decision maker is asked to accept such an extension.

We call the pair $(\precsim, \prec)$ *strengthened preference relation* if $\prec$ is not empty, moreover, we say that a function $U : \mathcal{L} \to \mathbb{R}$ *represents* (or *agrees with*) $(\precsim, \prec)$ if, for every $L, L' \in \mathcal{L}$

$$L \precsim L' \Rightarrow U(L) \leq U(L') \text{ and } L \prec L' \Rightarrow U(L) < U(L'). \tag{6}$$

In analogy with [4], given $(\precsim, \prec)$ on $\mathcal{L}$, our aim is to find a necessary and sufficient condition for the existence of a utility function $u : X \to \mathbb{R}$ such that the *Choquet expected utility* of g-lotteries in $\mathcal{L}$, defined for every $L \in \mathcal{L}$ as

$$\mathrm{CEU}(L) = \oint u \, \mathrm{d}Bel_L, \tag{7}$$

represents $(\precsim, \prec)$. In particular, since X is totally ordered by $\leq$ and $\mathrm{CEU}(\delta_{\{x\}}) = u(x)$ for every $x \in X$, we search for a *strictly increasing* u.

The next axiom requires that it is not possible to obtain two g-lotteries having the same aggregated basic assignment, by combining in the same way the aggregated basic assignments of two groups of g-lotteries, if each g-lottery in the first group is not preferred to the corresponding one in the second group, and at least a preference is strict.

Definition 2. *A strengthened preference relation $(\precsim, \prec)$ on a set $\mathcal{L}$ of g-lotteries is said to be* **Choquet rational** *if it satisfies the following condition:*

(g-CR) *For all $h \in \mathbb{N}$ and $L_i, L'_i \in \mathcal{L}$ with $L_i \precsim L'_i$ $(i = 1, \dots, h)$, if*

$$\mathbf{k}(M_{L_1}, \dots, M_{L_h}) = \mathbf{k}(M_{L'_1}, \dots, M_{L'_h})$$

with $\mathbf{k} = (k_1, \dots, k_h)$, $k_i > 0$ $(i = 1, \dots, h)$ and $\sum_{i=1}^{h} k_i = 1$, then it can be $L_i \prec L'_i$ for no $i = 1, \dots, h$. In particular, if $\precsim$ is complete, it must be $L_i \sim L'_i$ for every $i = 1, \dots, h$.

Note that the convex combination referred to in condition **(g-CR)** is the usual one involving probability distributions on X. Moreover, it is easily proven that if $\mathbf{k}(L_1, \dots, L_h) = \mathbf{k}(L'_1, \dots, L'_h)$, then it also holds $\mathbf{k}(M_{L_1}, \dots, M_{L_h}) = \mathbf{k}(M_{L'_1}, \dots, M_{L'_h})$ but the converse is generally not true.

The following theorem, proved in [2], shows that **(g-CR)** is a necessary and sufficient condition for the existence of a strictly increasing utility function u whose Choquet expected value on g-lotteries represents $(\precsim, \prec)$.

Theorem 1. *Let $\mathcal{L}$ be a finite set of g-lotteries, $X = \bigcup\{X_L : L \in \mathcal{L}\}$ with X totally ordered by $\leq$, and $(\precsim, \prec)$ a strengthened preference relation on $\mathcal{L}$. Assume $\mathcal{L}_0 \subseteq \mathcal{L}$ and for every $x, x' \in X$, $x \leq x'$ if and only if $\delta_{\{x\}} \precsim \delta_{\{x'\}}$. The following statements are equivalent:*

(i) *$(\precsim, \prec)$ is Choquet rational (i.e., it satisfies* **(g-CR)***);*
(ii) *there exists a strictly increasing function $u : X \to \mathbb{R}$ (unique up to a positive linear transformation), whose Choquet expected utility (CEU) on $\mathcal{L}$ represents $(\precsim, \prec)$.*

The proof of previous result provides an operative procedure to compute a strictly increasing utility function u on X in case **(g-CR)** is satisfied. For this, introduce the collections $S = \{(L_j, L'_j) : L_j \prec L'_j, L_j, L'_j \in \mathcal{L}\}$ and $R = \{(G_h, G'_h) : G_h \precsim G'_h, G_h, G'_h \in \mathcal{L}\}$ with $s = \operatorname{card} S$ and $r = \operatorname{card} R$. Then condition **(g-CR)** is equivalent to the *non-existence* of a row vector $\mathbf{k}$ of size $(1 \times s + r)$ with $k_i > 0$ for at least a pair $(L_i, L'_i) \in S$ and $\sum_{i=1}^{s+r} k_i = 1$ such that

$$\mathbf{k}(M_{L_1}, \dots, M_{L_s}, M_{G_1}, \dots, M_{G_r}) = \mathbf{k}(M_{L'_1}, \dots, M_{L'_s}, M_{G'_1}, \dots, M_{G'_r}).$$

In turn, setting $\mathbf{k} = (\mathbf{y}, \mathbf{z})$, previous condition is equivalent to the *non-solvability* of the following linear system (in which $||\cdot||_1$ denotes the L^1-norm)

$$\mathcal{S}' : \begin{cases} \mathbf{y}A + \mathbf{z}B = \mathbf{0} \\ \mathbf{y}, \mathbf{z} \geq \mathbf{0} \\ \mathbf{y} \neq \mathbf{0} \\ ||\mathbf{y}||_1 + ||\mathbf{z}||_1 = 1 \end{cases} \tag{8}$$

where $A = (a^j)$ and $B = (b^h)$ are, respectively, $(s \times n)$ and $(r \times n)$ real matrices with rows $a^j = M_{L'_j} - M_{L_j}$ for $j = 1, \dots, s$, and $b^h = M_{G'_h} - M_{G_h}$ for $h = 1, \dots, r$, and $\mathbf{y}$ and $\mathbf{z}$ are, respectively, $(1 \times s)$ and $(1 \times r)$ unknown row vectors.

By virtue of a well-known alternative theorem (see, e.g., [11]), in [2] the non-solvability of $\mathcal{S}'$ has been proven to be equivalent to the *solvability* of the following system

$$\mathcal{S} : \begin{cases} A\mathbf{w} > \mathbf{0} \\ B\mathbf{w} \geq \mathbf{0} \end{cases} \tag{9}$$

where $\mathbf{w}$ is a $(n \times 1)$ unknown column vector. In detail, setting $u(x_i) = w_i$, $i = 1, \dots, n$, the solution $\mathbf{w}$ induces a utility function u on X which, taking into account Remark 1, is strictly increasing and whose CEU represents $(\precsim, \prec)$.

3.1 A paradigmatic example

To motivate the topic dealt with in this paper we introduced the following example, which is inspired to the well-known Ellsberg's paradox [9].

Example 1. Consider the following hypothetical experiment. Let us take two urns, say U_1 and U_2, from which we are asked to draw a ball each. U_1 contains $\frac{1}{3}$ of white (w) balls and the remaining balls are black (b) and red (r), but in a ratio entirely unknown to us, analogously, U_2 contains $\frac{1}{4}$ of green (g) balls and the remaining balls are yellow (y) and orange (o), but in a ratio entirely unknown to us.

In light of the given information, the composition of U_1 singles out a class of probability measures $\mathbf{P}^1 = \{P^\theta\}$ on the power set $\wp(S_1)$ of $S_1 = \{w, b, r\}$ s.t. $P^\theta(\{w\}) = \frac{1}{3}$, $P^\theta(\{b\}) = \theta$, $P^\theta(\{r\}) = \frac{2}{3} - \theta$, with $\theta \in \left[0, \frac{2}{3}\right]$. Analogously, for the composition of U_2 we have the class $\mathbf{P}^2 = \{P^\lambda\}$ on $\wp(S_2)$ with $S_2 = \{g, y, o\}$ s.t. $P^\lambda(\{g\}) = \frac{1}{4}$, $P^\lambda(\{y\}) = \lambda$, $P^\lambda(\{o\}) = \frac{3}{4} - \lambda$, with $\lambda \in \left[0, \frac{3}{4}\right]$.

Concerning the ball drawn from U_1 and the one drawn from U_2, the following gambles are considered:

	w	b	r
L_1	100€	0€	0€
L_2	0€	0€	100€
L_3	0€	100€	100€
L_4	100€	100€	0€

	g	y	o
G_1	100€	10€	10€
G_2	10€	10€	100€
G_3	10€	100€	100€
G_4	100€	100€	10€

If we express the strict preferences $L_2 \prec L_1$, $L_4 \prec L_3$, then for no value of θ there exists a function $u : \{0, 100\} \to \mathbb{R}$ s.t. its expected value on the L_i's w.r.t. P^θ represents our preferences on the L_i's. Indeed, putting $w_1 = u(0)$ and $w_2 = u(100)$, both the following inequalities must hold $\frac{1}{3}w_1 + \theta w_1 + \left(\frac{2}{3} - \theta\right) w_2 < \frac{1}{3}w_2 + \theta w_1 + \left(\frac{2}{3} - \theta\right) w_1$ and $\frac{1}{3}w_2 + \theta w_2 + \left(\frac{2}{3} - \theta\right) w_1 < \frac{1}{3}w_1 + \theta w_2 + \left(\frac{2}{3} - \theta\right) w_2$, from which, summing memberwise, we get $w_1 + w_2 < w_1 + w_2$, i.e., a contradiction. The same can be proven if we express the strict preferences $G_2 \prec G_1$, $G_4 \prec G_3$.

Now take $\underline{P}^1 = \min \mathbf{P}^1$ and $\underline{P}^2 = \min \mathbf{P}^2$, where the minimum is intended pointwise on the elements of $\wp(S_1)$ and $\wp(S_2)$, obtaining:

$\wp(S^1)$	$\emptyset$	$\{w\}$	$\{b\}$	$\{r\}$	$\{w,b\}$	$\{w,r\}$	$\{b,r\}$	S_1
$\underline{P}^1$	0	$\frac{1}{3}$	0	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$	1

$\wp(S_2)$	$\emptyset$	$\{g\}$	$\{y\}$	$\{o\}$	$\{g,y\}$	$\{g,o\}$	$\{y,o\}$	S_2
$\underline{P}^2$	0	$\frac{1}{4}$	0	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	1

It is easily verified that both $\underline{P}^1$ and $\underline{P}^2$ are belief functions.

The gambles L_i's and G_i's allow to transport the belief functions $\underline{P}^1$ and $\underline{P}^2$ to the whole set of prizes $\{0,10,100\}$, obtaining the following g-lotteries with the corresponding aggregated basic assignments

	$\{0\}$	$\{10\}$	$\{100\}$	$\{0,10\}$	$\{0,100\}$	$\{10,100\}$	$\{0,10,100\}$
L_1	$\frac{2}{3}$	0	$\frac{1}{3}$	$\frac{2}{3}$	1	$\frac{1}{3}$	1
L_2	$\frac{1}{3}$	0	0	$\frac{1}{3}$	1	0	1
L_3	$\frac{1}{3}$	0	$\frac{2}{3}$	$\frac{1}{3}$	1	$\frac{2}{3}$	1
L_4	0	0	$\frac{1}{3}$	0	1	$\frac{1}{3}$	1
G_1	0	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	1	1
G_2	0	$\frac{1}{4}$	0	$\frac{1}{4}$	0	1	1
G_3	0	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{3}{4}$	1	1
G_4	0	0	$\frac{1}{4}$	0	$\frac{1}{4}$	1	1

	0	10	100
M_{L_1}	$\frac{2}{3}$	0	$\frac{1}{3}$
M_{L_2}	1	0	0
M_{L_3}	$\frac{1}{3}$	0	$\frac{2}{3}$
M_{L_4}	$\frac{2}{3}$	0	$\frac{1}{3}$
M_{G_1}	0	$\frac{3}{4}$	$\frac{1}{4}$
M_{G_2}	0	1	0
M_{G_3}	0	$\frac{1}{3}$	$\frac{3}{4}$
M_{G_4}	0	$\frac{3}{4}$	$\frac{1}{4}$

It is easily proven that for every strictly increasing $u:\{0,10,100\}\to\mathbb{R}$ the strict preferences $L_2 \prec L_1$, $L_4 \prec L_3$, $G_2 \prec G_1$, $G_4 \prec G_3$ are represented by their Choquet expected utility. Indeed, putting $w_1 = u(0)$, $w_2 = u(10)$, $w_3 = u(100)$, the following system

$$\mathcal{S}: \begin{cases} w_1 < \frac{2}{3}w_1 + \frac{1}{3}w_3 \\ \frac{2}{3}w_1 + \frac{1}{3}w_3 < \frac{1}{3}w_1 + \frac{2}{3}w_3 \\ w_2 < \frac{3}{4}w_2 + \frac{1}{4}w_3 \\ \frac{3}{4}w_2 + \frac{1}{4}w_3 < \frac{1}{4}w_2 + \frac{3}{4}w_3 \\ w_1 < w_2 < w_3 \end{cases}$$

is such that any choice of values satisfying $w_1 < w_2 < w_3$ is a solution.

Now, suppose to toss a fair coin and to choose among L_1 and G_1 depending on the face shown by the coin. In analogy, suppose to choose among L_2 and G_1 with a totally similar experiment. Let us denote with F_1 and F_2 the results of the two experiments. This implies that F_1 and F_2 can be defined as the convex combinations $F_1 = \frac{1}{2}L_1 + \frac{1}{2}G_1$ and $F_2 = \frac{1}{2}L_2 + \frac{1}{2}G_1$, obtaining the g-lotteries with the corresponding aggregated basic assignments

	$\{0\}$	$\{10\}$	$\{100\}$	$\{0,10\}$	$\{0,100\}$	$\{10,100\}$	$\{0,10,100\}$
F_1	$\frac{8}{24}$	$\frac{9}{24}$	$\frac{7}{24}$	$\frac{17}{24}$	$\frac{15}{24}$	$\frac{16}{24}$	1
F_2	$\frac{4}{24}$	$\frac{9}{24}$	$\frac{3}{24}$	$\frac{13}{24}$	$\frac{15}{24}$	$\frac{12}{24}$	1

	0	10	100
M_{F_1}	$\frac{8}{24}$	$\frac{9}{24}$	$\frac{7}{24}$
M_{F_2}	$\frac{12}{24}$	$\frac{9}{24}$	$\frac{3}{24}$

If we add to previous preferences the further strict preference $F_1 \prec F_2$ then there is no strictly increasing $u:\{0,10,100\}\to\mathbb{R}$ whose Choquet expected utility represents our preferences. Indeed, in this case, the extended system

$$\mathcal{S}: \begin{cases} w_1 < \frac{2}{3}w_1 + \frac{1}{3}w_3 \\ \frac{2}{3}w_1 + \frac{1}{3}w_3 < \frac{1}{3}w_1 + \frac{2}{3}w_3 \\ w_2 < \frac{3}{4}w_2 + \frac{1}{4}w_3 \\ \frac{3}{4}w_2 + \frac{1}{4}w_3 < \frac{1}{4}w_2 + \frac{3}{4}w_3 \\ \frac{8}{24}w_1 + \frac{9}{24}w_2 + \frac{7}{24}w_3 < \frac{12}{24}w_1 + \frac{9}{24}w_2 + \frac{3}{24}w_3 \\ w_1 < w_2 < w_3 \end{cases}$$

admits no solution. Notice that, taking into account Remark 1, condition **(g-CR)** fails since it holds

$$\frac{3}{4}M_{F_1} + \frac{1}{8}M_{\delta_{\{0\}}} + \frac{1}{8}M_{\delta_{\{10\}}} = \frac{3}{4}M_{F_2} + \frac{1}{8}M_{\delta_{\{10\}}} + \frac{1}{8}M_{\delta_{\{100\}}}.$$

3.2 Extension of Choquet rational preferences

In previous section it has been shown that condition **(g-CR)** is equivalent to the existence of a strictly increasing utility function u on X, whose CEU represents $(\precsim, \prec)$, moreover, such a u can be explicitly determined by solving the linear system $\mathcal{S}$ defined in (9). It is straightforward that once a utility u has been fixed, a complete preference relation on $\mathcal{L}$ (or any finite superset $\mathcal{L}'$ of g-lotteries on the same finite set X) extending $(\precsim, \prec)$ is induced by the corresponding CEU functional.

Nevertheless, system $\mathcal{S}$ has generally infinite solutions which can give rise to possibly very different complete preference relations, thus any choice of a utility function causes a loss of information, moreover, it is not clear why one should choose a utility function in place of another.

This is why it is preferable to face the extension in a qualitative setting by considering the entire class of utility functions whose CEU represents the preference $(\precsim, \prec)$ and suggesting to the decision maker those pairs of g-lotteries where all the utility functions unanimously agree. In this view, the following Theorem 2 proves the extendibility of a Choquet rational relation and shows how condition **(g-CR)** guides the decision maker in assessing his preferences.

Theorem 2. *Let X be a finite set totally ordered by $\leq$, $\mathcal{L}$ and $\mathcal{L}'$ finite sets of g-lotteries on X, with $\mathcal{L} \subseteq \mathcal{L}'$, and $(\precsim, \prec)$ a strengthened preference relation on $\mathcal{L}$. Assume $\mathcal{L}_0 \subseteq \mathcal{L}$ and for every $x, x' \in X$, $x \leq x'$ if and only if $\delta_{\{x\}} \precsim \delta_{\{x'\}}$. Then if $(\precsim, \prec)$ satisfies condition* **(g-CR)** *there exists a family $\{\precsim^\gamma : \gamma \in \mathit{\Gamma}\}$ of complete relations on $\mathcal{L}'$ satisfying* **(g-CR)** *which extend $(\precsim, \prec)$. Moreover, denoting with $\prec^\gamma$ and $\sim^\gamma$, respectively, the strict and symmetric parts of $\precsim^\gamma$, for $\gamma \in \mathit{\Gamma}$, condition* **(g-CR)** *singles out the relations*

$$\prec^\star = \bigcap\{\prec^\gamma : \gamma \in \mathit{\Gamma}\} \quad \textit{and} \quad \sim^\star = \bigcap\{\sim^\gamma : \gamma \in \mathit{\Gamma}\}.$$

Proof. Suppose $X = \{x_1, \ldots, x_n\}$ with $x_1 < \ldots < x_n$. By the proof of Theorem 1 (see [2]), $(\precsim, \prec)$ satisfies condition **(g-CR)** if and only if system $\mathcal{S}$ defined in (9) admits a $(n \times 1)$ column vector $\mathbf{w}$ as solution. In turn, setting $u(x_i) = w_i$, for $i = 1, \ldots, n$, we get a strictly increasing utility function u on X whose Choquet expected value represents $(\precsim, \prec)$ on $\mathcal{L}$. Defining for every $L, L' \in \mathcal{L}'$

$$L \precsim^\gamma L' \Leftrightarrow \mathrm{CEU}(L) \leq \mathrm{CEU}(L'),$$

we get a relation $\precsim^\gamma$ on $\mathcal{L}'$ which is complete and satisfies **(g-CR)** by virtue of Theorem 1. This implies that the family $\{\precsim^\gamma : \gamma \in \mathit{\Gamma}\}$ is not empty and all its members are obtained varying the solution $\mathbf{w}$ of system $\mathcal{S}$. The correspondence between the set of solutions and the family of relations $\{\precsim^\gamma : \gamma \in \varGamma\}$ is onto but not one-to-one, as every positive linear transformation of a solution $\mathbf{w}$ gives rise to the same relation $\precsim^\gamma$.

The relations $\prec^\star$ and $\sim^\star$ express, respectively, the pairs of g-lotteries in $\mathcal{L}'$ on which all the strict $\prec^\gamma$ and symmetric $\sim^\gamma$ parts, for $\gamma \in \varGamma$, agree. It trivially holds that $\prec^\star$ and $\sim^\star$ extend the relations $\prec$ and $\sim$ obtained from $(\precsim, \prec)$, moreover, in order to determine $\prec^\star$ and $\sim^\star$, for every $F, G \in \mathcal{L}'$ such that it does not hold $F \prec G$ or $G \prec F$ or $F \sim G$ it is sufficient to test the solvability of the three linear systems

$$\mathcal{S}^{\prec^\star} : \begin{cases} A'\mathbf{w} > \mathbf{0} \\ B\mathbf{w} \geq \mathbf{0} \end{cases} \qquad \mathcal{S}^{\succ^\star} : \begin{cases} A''\mathbf{w} > \mathbf{0} \\ B\mathbf{w} \geq \mathbf{0} \end{cases} \qquad \mathcal{S}^{\sim^\star} : \begin{cases} A\mathbf{w} > \mathbf{0} \\ B'\mathbf{w} \geq \mathbf{0} \end{cases}$$

where $\mathbf{w}$ is an unknown $(n \times 1)$ column vector, A and B are, respectively, $(s \times n)$ and $(r \times n)$ real matrices defined as in (8), A' is a $((s+1) \times n)$ real matrix obtained adding to A the $(s+1)$-th row $a^{(s+1)} = M_G - M_F$, A'' is a $((s+1) \times n)$ real matrix obtained adding to A the $(s+1)$-th row $a^{(s+1)} = M_F - M_G$, and B' is a $((r+2) \times n)$ real matrix obtained adding to B the $(r+1)$-th row $b^{(r+1)} = M_G - M_F$ and the $(r+2)$-th row $b^{(r+2)} = M_F - M_G$.

Depending on the solvability of systems $\mathcal{S}^{\prec^\star}, \mathcal{S}^{\succ^\star}, \mathcal{S}^{\sim^\star}$ we can have the following situations:

(a) $F \prec^\star G$ if and only if $\mathcal{S}^{\prec^\star}$ is solvable and $\mathcal{S}^{\succ^\star}, \mathcal{S}^{\sim^\star}$ are not, as this happens if and only if $\mathrm{CEU}(F) < \mathrm{CEU}(G)$ for every u given by a solution of $\mathcal{S}$;
(b) $G \prec^\star F$ if and only if $\mathcal{S}^{\succ^\star}$ is solvable and $\mathcal{S}^{\prec^\star}, \mathcal{S}^{\sim^\star}$ are not, as this happens if and only if $\mathrm{CEU}(G) < \mathrm{CEU}(F)$ for every u given by a solution of $\mathcal{S}$;
(c) $F \sim^\star G$ if and only if $\mathcal{S}^{\sim^\star}$ is solvable and $\mathcal{S}^{\prec^\star}, \mathcal{S}^{\succ^\star}$ are not, as this happens if and only if $\mathrm{CEU}(F) = \mathrm{CEU}(G)$ for every u given by a solution of $\mathcal{S}$.

In all the remaining cases, the Choquet expected utilities determined by solutions of $\mathcal{S}$ do not unanimously agree in ordering the pair F and G. □

Relations $\prec^\star$ and $\sim^\star$ determined in the proof of previous theorem express "forced" preferences that the decision maker has to accept in order to maintain Choquet rationality. On the other hand, pairs of g-lotteries not ruled by $\prec^\star$ and $\sim^\star$ are subject to a choice by the decision maker. In the latter situation, a subjective elicitation is required or, in case of a software agent [17], a suitable automatic choice criterion can be adopted.

We stress that each choice made by the decision maker imposes a new constraint in system $\mathcal{S}$, thus the set of utility functions whose CEU represents the current strengthened preference $(\precsim, \prec)$ is possibly reduced.

Previous discussion suggests the following Algorithm 1 which is thought to guide the decision maker in enlarging a Choquet rational preference relation $(\precsim, \prec)$ to a (possibly new) pair of g-lotteries F and G: the extended preference is still denoted as $(\precsim, \prec)$. In particular, Algorithm 1 returns to the decision maker what he must do or he cannot do in order to maintain **(g-CR)**.

Notice that possibly $F, G \in \mathcal{L}$, thus previous algorithm can be used to produce a step by step completion of the preference relation $(\precsim, \prec)$ on $\mathcal{L}$.

Algorithm 1 Extension of a Choquet rational relation

```
function EXTENSION(($\precsim, \prec$), $F$, $G$)
    if $\mathcal{S}^{\prec^\star}$ and $\mathcal{S}^{\succ^\star}$ are solvable then free preference between $F$ and $G$
    else if $\mathcal{S}^{\prec^\star}$ is solvable and $\mathcal{S}^{\sim^\star}$ is not then it must be $F \prec G$
    else if $\mathcal{S}^{\succ^\star}$ is solvable and $\mathcal{S}^{\sim^\star}$ is not then it must be $G \prec F$
    else if $\mathcal{S}^{\prec^\star}$ and $\mathcal{S}^{\sim^\star}$ are solvable then it cannot be $G \prec F$
    else if $\mathcal{S}^{\succ^\star}$ and $\mathcal{S}^{\sim^\star}$ are solvable then it cannot be $F \prec G$
    else it must be $F \sim G$
end function
```

Algorithm 1 requires as input a Choquet rational preference relation $(\precsim, \prec)$ on a set of g-lotteries $\mathcal{L}$, and two (possibly new) g-lotteries F and G, all rewritten on $X = \{x_1, \ldots, x_n\}$ with $x_1 < \ldots < x_n$. The g-lotteries in $\mathcal{L} \cup \{F, G\}$ can be simply regarded as basic assignments on $\wp(X)$, i.e., as real $(1 \times q)$ row vectors with $q = 2^n - 1$. The formation of matrices A, A', A'', B, B' requires the computation of the aggregated basic assignment M_L for every $L \in \mathcal{L} \cup \{F, G\}$, which can be done in polynomial time with respect to q.

The extension is faced through the solution of at most three linear programming problems, whose solution has time complexity which is a polynomial in $n = \log_2(q+1)$ and the digital size of the coefficients in matrices A', B or A'', B or A, B', respectively [16].

The following example shows an application of Algorithm 1.

Example 2. Consider the situation described in Example 1. It has already been observed that adding the further strict preference $F_1 \prec F_2$ implies that the global preference relation has no more a Choquet expected utility representation. We use Algorithm 1 to guide the decision maker in judging his preference between F_1 and F_2 in order to preserve Choquet rationality. It is easily seen that only system

$$\mathcal{S}^{\succ^\star} : \begin{cases} w_1 < \frac{2}{3}w_1 + \frac{1}{3}w_3 \\ \frac{2}{3}w_1 + \frac{1}{3}w_3 < \frac{1}{3}w_1 + \frac{2}{3}w_3 \\ w_2 < \frac{3}{4}w_2 + \frac{1}{4}w_3 \\ \frac{3}{4}w_2 + \frac{1}{4}w_3 < \frac{1}{4}w_2 + \frac{3}{4}w_3 \\ \frac{12}{24}w_1 + \frac{9}{24}w_2 + \frac{3}{24}w_3 < \frac{8}{24}w_1 + \frac{9}{24}w_2 + \frac{7}{24}w_3 \\ w_1 < w_2 < w_3 \end{cases}$$

is solvable while $\mathcal{S}^{\prec^\star}$ and $\mathcal{S}^{\sim^\star}$ are not. In turn, this implies that $F_2 \prec^\star F_1$ and so the decision maker is forced to strictly prefer F_1 to F_2 to respect condition **(g-CR)**.

On the other hand, considering the g-lotteries L_1 and G_1, both systems

$$\mathcal{S}^{\prec^\star} : \begin{cases} w_1 < \frac{2}{3}w_1 + \frac{1}{3}w_3 \\ \frac{2}{3}w_1 + \frac{1}{3}w_3 < \frac{1}{3}w_1 + \frac{2}{3}w_3 \\ w_2 < \frac{3}{4}w_2 + \frac{1}{4}w_3 \\ \frac{3}{4}w_2 + \frac{1}{4}w_3 < \frac{1}{4}w_2 + \frac{3}{4}w_3 \\ \frac{2}{3}w_1 + \frac{1}{3}w_3 < \frac{3}{4}w_2 + \frac{1}{4}w_3 \\ w_1 < w_2 < w_3 \end{cases} \qquad \mathcal{S}^{\succ^\star} : \begin{cases} w_1 < \frac{2}{3}w_1 + \frac{1}{3}w_3 \\ \frac{2}{3}w_1 + \frac{1}{3}w_3 < \frac{1}{3}w_1 + \frac{2}{3}w_3 \\ w_2 < \frac{3}{4}w_2 + \frac{1}{4}w_3 \\ \frac{3}{4}w_2 + \frac{1}{4}w_3 < \frac{1}{4}w_2 + \frac{3}{4}w_3 \\ \frac{3}{4}w_2 + \frac{1}{4}w_3 < \frac{2}{3}w_1 + \frac{1}{3}w_3 \\ w_1 < w_2 < w_3 \end{cases}$$

are solvable, thus in this case the decision maker is totally free to choose his preference between L_1 and G_1.

References

1. Chateauneuf, A., Cohen, M.: Choquet expected utility model: a new approach to individual behavior under uncertainty and social choice welfare. *Fuzzy Meas. and Int.: Th. and App.*, pp. 289–314, Heidelberg: Physica (2000).
2. Coletti, G., Petturiti, D., Vantaggi, B.: Choquet expected utility representation of preferences on generalized lotteries. *IPMU 2014*, *Part II*, *CCIS 443*, A. Laurent et al. (Eds.), pp. 444–453 (2014).
3. Coletti, G., Petturiti, D., Vantaggi, B.: Rationality principles for preferences on belief functions. Submitted to *Kybernetika*.
4. Coletti, G., Regoli, G.: How can an expert system help in choosing the optimal decision?. *Th. and Dec.*, 33(3), 253–264 (1992).
5. Coletti, G., Scozzafava, R.: Toward a General Theory of Conditional Beliefs. *Int. J. of Int. Sys.*, 21, 229–259 (2006).
6. Coletti, G., Scozzafava, R., Vantaggi, B.: Inferential processes leading to possibility and necessity. *Inf. Sci.*, 245, 132–145 (2013).
7. Dempster, A.P.: Upper and Lower Probabilities Induced by a Multivalued Mapping. *Ann. of Math. Stat.*, 38(2), 325–339 (1967).
8. Denneberg, D.: *Non-additive Measure and Integral.* Theory and Decision Library: Series B, Vol. 27, Kluwer Academic, Dordrecht, Boston (1994).
9. Ellsberg, D.: Risk, Ambiguity and the Savage Axioms. *Quart. Jour. of Econ.*, 75, 643–669 (1961).
10. Fagin, R., Halpern, J.Y.: Uncertainty, belief and probability. *Comput. Int.*, 7(3), 160–173 (1991).
11. Gale, D.: *The Theory of Linear Economic Models.* McGraw Hill (1960).
12. Gilboa, I., Schmeidler, D.: Additive representations of non-additive measures and the Choquet integral. *Ann. of Op. Res.*, 52, 43–65 (1994).
13. Mc Cord, M., de Neufville, R.: Lottery Equivalents: Reduction of the Certainty Effect Problem in Utility Assessment. *Man. Sci.*, 23(1), 56–60 (1986).
14. Miranda, E., de Cooman, G., Couso, I.: Lower previsions induced by multi-valued mappings. *J. of Stat. Plan. and Inf.*, 133, 173–197 (2005).
15. Quiggin, J.: A Theory of Anticipated Utility. *J. of Ec. Beh. and Org.*, 3, 323–343 (1982).
16. Papadimitriou, C.H., Steiglitz, K.: *Combinatorial Optimization: Algorithms and Complexity.* Dover, New York (1998).
17. Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach.* Second edition. Prentice Hall, Upper Saddle River (2003).
18. Savage, L.: *The foundations of statistics.* Wiley, New York (1954).
19. Shafer, G.: *A Mathematical Theory of Evidence.* Princeton University Press (1976).
20. Schmeidler, D.: Subjective probability and expected utility without additivity. *Econometrica*, 57(3), 571–587 (1989).
21. Schmeidler, D.: Integral representation without additivity. *Proc. of the Am. Math. Soc.*, 97(2), 255–261, 1986.
22. Smets, P.: Decision making in the TBM: the necessity of the pignistic transformation. *Int. J. Approx. Reas.*, 38(2), 133–147 (2005).
23. von Neumann, J., Morgenstern, O.: *Theory of Games and Economic Behavior.* Princeton University Press (1944).

On multiple learning schemata in conflict driven solvers

Andrea Formisano[1] and Flavio Vella[2]

[1] Università di Perugia formis@dmi.unipg.it
[2] IAC-CNR and Sapienza Università di Roma vella@di.uniroma1.it

Abstract. In this preliminary paper we describe a general approach for multiple learning in conflict-driven SAT solvers. The proposed formulation of the conflict analysis task turns out to be expressive enough to reckon with different orthogonal generalizations of the standard learning schemata, such as the conjunct analysis of multiple conflicts, the generation of possibly interdependent learned clauses, the imposition of global optimality criteria.
We formalize the general learning problem as a search for a collection of vertex cuts in a directed acyclic graph. Optimality of the solution may be evaluated with respect to a given global objective function intended to encode search strategies and heuristics affecting the behavior of the solver. We also outline some algorithmical solutions by exploiting standard algorithms proposed to solve cut and multicut problems on DAGs.

1 Introduction

Most of the successful SAT solvers available nowadays originate from refinements of the DPLL procedure [11] and integrate powerful techniques such as *conflict driven learning* and non-chronological backtracking. As a matter of fact, the combination of suitable learning schemata with smart branching heuristics and efficient (Boolean) constraint propagation algorithms [4], remarkably improved the efficiency and effectiveness of modern SAT solvers. Analogous techniques, often migrated from SAT technology, have been exploited in developing solvers in various fields of Automated Reasoning, such as Answer Set Programming, Constraint Programming, and Satisfaction Modulo Theory (cf., among many, [13, 26, 23] and the references therein).

In what follows we focus on clause-based SAT solving, albeit similar arguments can be advanced concerning other kind of solvers. More specifically, we will consider the problem of determining the satisfiability of a set of clauses, built up from a collection of propositional variables. (As usual, a literal is a variable or its complement. Clauses are sets of literals.)

Let us briefly recall the main traits of a DPLL-like SAT solver. For a formal and detailed treatment the reader is referred to [4], among many). Given an instance of SAT (i.e., a set of clauses), a DPLL-like SAT solver proceeds by alternating decision steps and propagation phases. By making a decision, the solver assigns one propositional variable a truth value. Then, it propagates the effects of such a decision to (possibly) derive implied assignments. Each decision has a *decision level* associated to it and propagation takes place, within the current decision level, whenever all but one literals in a clause have been assigned false (with a slight abuse, let us call *unit* this kind of clause).

In order to find a satisfying assignment for the unit clause (and then, for the whole instance), the unassigned literal must be set true. The propagation stage continues as long as units are produced. Then, the decision level is increased and another decision is taken. The process stops as soon as a solution is found (namely, all variables have been consistently assigned) or a *conflict* is detected. Normally, a conflict arises when, through the propagation phase, all the literals in a clause become assigned to false. At this point a *conflict analysis* procedure derives a new *conflict clause* to be added to the instance. Then, some of the previously taken decisions are undone and the solver *backjumps* to a previous decision level, before continuing the search for a solution. The presence of the new clause drives subsequent propagation phases and prevents the solver from generating again the very same conflicting assignment.

In this paper we propose a schema for conflict analysis general enough to enable the conjunct analysis of several conflicts and the consequent generation of more that one learned clause. Multiplicity may arise not only from generating more than one conflict in a single propagation phase, or by admitting multiple decisions at each decision level, but also from concurrently running different instances of the above outlined procedure (each one performing a different visit of the solution space). Designing a global learning procedure in such a general context has several potential advantages. On the one hand, it might take advantage from the results of all searches in order to derive more effective conflict clauses. On the other hand, learned clauses convey knowledge exchange between the concurrent threads of the parallel solver.

The paper is organized as follows. After recalling the basic notions about conflict-driven SAT solving (Section 2), in Section 3 we formalize our general learning schema. Sections 4 and 5 concretize our proposal by introducing some algorithmical solutions. Finally, Section 6 provides some concluding remarks and hints for future development.

2 Conflict analysis and implication graphs

Let us consider in more detail the conflict analysis procedure described earlier. Following [34, 22], the dependencies between decided and propagated variables can be described by means of an *implication graph*. It is a directed acyclic graph (DAG, for short) in which vertices represent truth value assignments for literals. We will often identify a vertex with the literal it represents: a variable x assigned true (resp., false) is rendered by a vertex x (resp., $-x$). Moreover, given a literal x, let $\overline{x}$ denote its complement.

Edges express the reasons that lead the assignments. In particular, decided variables correspond to vertices having no incident edges. If a literal x has been assigned true by propagation, because of a unit clause $\{x, y_1, \ldots, y_k\}$, then the vertex x has each $\overline{y_i}$ as direct antecedents in the implication graph.

To simplify the following treatment, let us introduce a special kind of vertex. A *conflict vertex*, not corresponding to any variable, is introduced in the graph whenever a pair of contradictory assignments is produced for the same propositional variable x. A conflict vertex has exactly two antecedent vertices, representing two inconsistent value assignments for a variable.

In this setting, given an implication graph $G(V, E)$, we can identify the set of *source* vertices $S \subseteq V$ (corresponding to decisions) and the set of conflict vertices $T \subseteq V$.

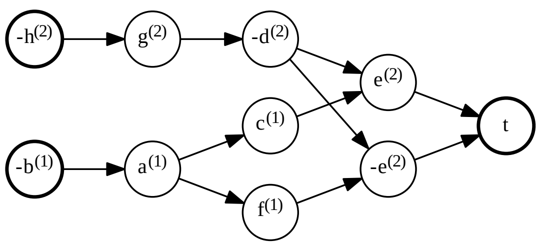


Fig. 1. Implication graph G_1 for Example 1.

Example 1. Consider the following clauses: $\{a, b\}$, $\{c, \neg a\}$, $\{\neg a, f\}$, $\{h, g\}$, $\{\neg g, \neg d\}$, $\{\neg f, d, \neg e\}$, $\{e, d, \neg c\}$. Fig. 1 shows an implication graph G_1 obtained by assigning true the two literals $\neg b$ and $\neg h$. Hence, two sources are present in G_1, corresponding to such decisions. (For the time being, let us ignore the decision levels indicated by the vertices' superscripts.) As mentioned, edges denote propagation steps. For instance, an edge from $\neg b$ to a is introduced because of the first clause. In fact, being b false, in order to satisfy the clause, a must be set true. Similarly, two edges from a, to vertices c and f, respectively, are introduced because of the second and the third clause, resp., and so on. Note that, once completed, the propagation steps introduces a pair of conflicting assignments for the literal e. A conflict vertex t denotes this fact. $\square$

For simplicity, let us consider a graph G having a single conflict vertex t (having x and $\overline{x}$ as antecedents, for a variable x).[1] Every vertex cut X in G that separates S from t and such that $\{x, \overline{x}\} \not\subseteq X$, corresponds to a partial assignment sufficient to imply the conflict.[2] In other words, X can be translated into a conflict clause made of the complements of the literals in X.

Example 2. Consider the graph G_1 in Fig. 1. The vertex cut made of the vertices a and g separates all the sources from the conflict. Consequently, a partial assignment sufficient to imply the conflict consists in assigning true to both a and g.

Clearly, more that one vertex cut may exist. For instance, in G_1, two other possible vertex cuts are $\{a, \neg d\}$ and $\{c, f, \neg d\}$. $\square$

If a vertex cut contains only one literal that has been assigned at a certain decision level ℓ, then such a literal is a *unique implication point* (UIP). Notice that, if s_ℓ is the literal decided at level ℓ, a UIP *dominates* (in the sub-graph of level ℓ) the conflict vertex t with respect to s_ℓ: each path from the source s_ℓ to t must go through the UIP.

The maximum decision level of the vertices in the cut (except the decision level of the conflict) is the decision level to backjump. After backjumping, the conflict clause becomes an *asserting clause*: all its literals but one are assigned false, so the remaining

[1] Such a situation can be always achieved by isolating a single conflict vertex t, and by restricting the implication graph to the *S–t-connected* portion of G.

[2] Let a and b be two distinct vertices such that b is reachable from a in a directed graph G. Recall that, a *vertex cut* (resp., *edge cut*) that separates a from b, is defined as a set of vertices (resp., edges) such that their removal from G eliminates all directed paths from a to b.

literal is determined by propagation. Consequently, the solver will be "guided" toward a different portion of the search space.

With respect to the same conflict, several UIPs may occur in the implication graph. These are ordered depending on the distance from the conflict vertex: the UIP closest to the conflict vertex is called *first UIP*, and so on.

Clearly, not all the possible vertex cuts involve a UIP. Nevertheless, each of them identifies a different conflict clause that can be, in principle, profitably added to the SAT instance in order to prune the search space.

Example 3. Consider again the graph G_1 in Fig. 1 and assume that $\neg b$ has been assigned first, at the decision level 1. (Decision levels are denoted by superscripts in the graph.) All the vertices assigned by propagation as a consequence of this decision belong to such decision level. Then, a subsequent decision, at level 2, assigned true to the literal $\neg h$ and caused the propagation of the other vertices. Hence, the conflict occurs at level 2. The first UIP is $\neg d$. Indeed it dominates t (each path from $\neg h$ to the conflict must go through the vertex $\neg d$) and it is the dominator closest to t. The procedure proceeds by learning the clause corresponding to such UIP, namely $\{\neg f, \neg c, d\}$, and backjumping to level 1 (hence, the decision regarding $\neg h$ is undone). In this situation, a propagation step, using the newly introduced clause forces d to be assigned true. In turn, because of clauses $\{\neg g, \neg d\}$ and $\{h, g\}$ also $\neg g$ and h are set true. □

3 Generalizing conflict analysis and learning

The effectiveness of a solver largely depends on the strategy used to identify suitable conflict clauses, among all the possible candidates, by analyzing the implication graph. Notice that, the structure of the implication graph substantially affects the learning procedure, because, as mentioned, conflict clauses correspond to vertex cuts. In turn, the structure of the graph depends on:

(a) the selection of decision variables (branching strategy). This determines the set of sources of the graph.

(b) The specific sequence of propagations performed in each propagation phase. If different asserting clauses are activable at each time, different orders of their activation induce different topologies of the graph.

Many of the existing solvers perform just a single decision at each step and stop the propagation phase as soon as the first conflict arises. Moreover, conflict analysis is usually focused on UIPs (often, only conflict clauses involving the first-UIP are learned). One reason for this choice is that such conflict clauses can be obtained by developing a linear derivation by propositional resolution. The steps of this derivation use as resolvents (in the reverse order) the asserting clauses used along the path from the (first) UIP to the conflict. Hence, in some sense, once the conflict is present, the generation of the conflict clause is deterministic [4, 34].

In what follows we generalize the basic learning schema described so far. More specifically, we design a general schema for conflict analysis that takes into account the following aspects:

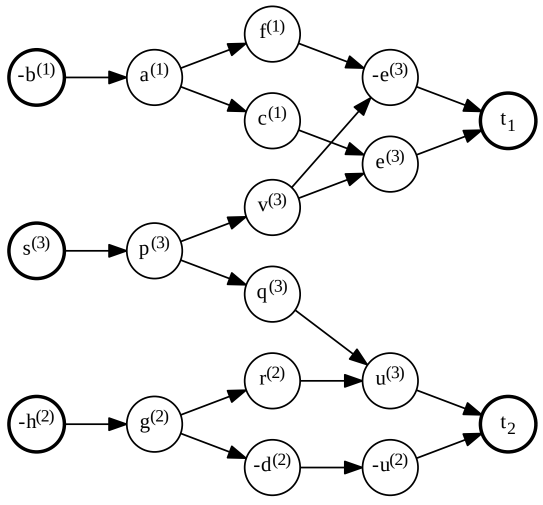


Fig. 2. Implication graph G_2 for Example 4.

- Multiple decisions taken at each decision level. This corresponds to focusing the search for a solution on a more restricted portion of the search space. This increases the effect of subsequent propagation phases. In addition, this splitting of the search space may be exploited by parallel solvers.
- Multiple conflicts. These might be generated both because of multiple decisions or because the propagation phase does not necessarily end when the first conflict is produced. For example, it might proceed until no further inference is possible.
- Weights associated to vertices and/or edges of the implication graph. This enables the application of some kind of heuristics in selecting conflict clauses. We will not enter into the details of how these weights are assigned. For the time being, it suffices observing that weights may be generated in various manners, depending on the involved heuristics. For instance, one could consider static measures, concerning the structure of the SAT instance at hand, or dynamic parameters such as the "relevance" exhibited by each variable during the previous part of the computation (as an immediate example, think about exploiting criteria somewhat akin to those used by branching strategies).

Most of the learning schemata appeared in the literature essentially consider only cuts involving UIPs. In this context, some comparative empirical evaluation of different learning schemata can be found in [34] and in [27]. Very few proposals concern multiple clauses learning. We just mention here the interesting work [17], that explores the advantage of learning several clauses from the same conflict.

Example 4. Consider this set of clauses: $\{a, b\}$, $\{c, \neg a\}$, $\{\neg a, f\}$, $\{h, g\}$, $\{\neg g, \neg d\}$, $\{e, \neg v, \neg c\}$, $\{\neg f, \neg v, \neg e\}$, $\{q, \neg p\}$, $\{v, \neg p\}$, $\{r, \neg g\}$, $\{d, \neg u\}$, $\{p, \neg s\}$, $\{u, \neg r, \neg q\}$. Suppose we proceed by deciding a single literal in each decision level and we develop an implication graph by first deciding $\neg b$ to be set true (decision level 1). After propagating the effect of this decision (literals a, c, f are set true, cf., the implication graph G_2 in Fig. 2), we step to decision level 2 and set true the literal $\neg h$. Again, a propagation phase is executed and this determines the truth values of the literals $g, \neg d, r$, and $\neg u$. Since no conflict arises and there are still unassigned literals, we step to decision level 3. Suppose we perform a decision setting true the literal s. Fig. 2 show the literals propagated at level 3, namely, $p, q, v, \neg e, e$ and u. Clearly, two conflicts (denoted by the conflict vertices, t_1 and t_2) arise, because of the values to be assigned to u and e, respectively.

The cut corresponding to the first UIP for the conflict t_1 is made of the vertices v, f, and c. The clause learned from this conflict is $\{\neg v, \neg f, \neg c\}$. Similarly, by analyzing the other conflict t_2, we obtain a cut made of the vertices $\neg d$, r, and q, corresponding to the first UIP q, and an alternative learned clause $\{d, \neg r, \neg q\}$..

A solver that decides a single literal in each decision level and that takes into account only one conflict, would learn just one of these two clauses, while both of them can be added to the set of clauses and affect the subsequent part of the execution.

Moreover, performing a global analysis of the implication graph might help in learning alternative clauses. (Note that, the same graph can be obtained by deciding the three literals $\neg b, s, \neg h$ at the first decision level, except for the fact that in that case all vertices belong to level 1.) For instance, the pair of clauses $\{\neg a, \neg p\}$ and $\{\neg g, \neg p\}$ can be obtained by considering the two sets $X_1 = \{a, p\}$ and $X_2 = \{p, g\}$, each of them separating one of the conflicts from all source vertices $\neg b$, $\neg h$, and s. In a situation where one heuristically prefers short clauses, the global analysis produces better results. Observe, moreover, that detecting a single vertex cut separating both conflict vertices from all sources would produce a single learned clause. However, such a clause would contain at least three literals. □

The previous example shows that, even considering simple heuristics based on cardinality of learned clauses (i.e., the number of literals they include), a global analysis may offer advantages. In general, more complex heuristics can be applied to assign weights to literals/vertices and then to vertex cuts. Consequently, the conflict analysis procedure will focus on minumin vertex cuts. Among the various approaches that can be adopted in assigning weights to literals, many of the branching strategies exploited in standard SAT-solvers can be adapted to our case [4].

Remark 1. Note that another generalization can be foreseen. Indeed, it seems natural to consider the parallel execution of several solvers, each one adopting its own criteria and heuristics, and exploring (possibly) different portions of the search space. Plainly, each solver develops a different implication graph. A conjunct analysis of all these graphs (that share the vertices and differ in the set of edges) should be advisable. Intuitively, a common analysis could be enabled by introducing a coloring of the edges. Each color would identify the inferences made by one of the solver. In this way, a global learning process can be developed, by applying global strategies, so as to realize forms of communication and cooperation between the solvers. For the sake of simplicity, in what follows we will not deal with this kind of generalization, which represents an interesting theme for future research.

Summing up, we will consider implication graphs where weights are provided and several conflict vertices may occur. The graphs are still acyclic and layered (as vertices are partitioned by the decision levels). The conflict analysis should focus on those (sets of) cuts that are optimal with respect to a given objective function. Typically, it may encode "local" optimality criteria (focusing on each single conflict in isolation from the others), as well as involve "global" criteria, aimed at optimizing the set of cuts as a whole. To keep the following description as general as possible, we will leave these optimality criteria implicit and simply deal with a generic objective function $f(\cdot)$.

Let $G = (V, E)$ be a weighted DAG with $n = |V|$ vertices and $m = |E|$ edges. Let $S \subseteq V$ be the set of source vertices and $T = \{t_1, \ldots, t_k\} \subseteq (V - S)$ the set of sink vertices. Assume, moreover, that each sink is reachable (through a directed path) from at least one source.

Without loss of generality, we can assume that $S = \{s_1, \ldots, s_k\}$ and that t_i is reachable from s_i (for each i). [3]

A weight function $w : V \to \mathbb{N}$ assigns positive weights to the vertices in G.[4] We are interested in solving the following optimization problem (where $f(\cdot)$ is the objective function).

Problem 1 *Find k vertex sets $X_i \subseteq (V - T)$ that minimize the value of $f(X_1, \ldots, X_k)$ and such that, for each $i \in \{1, \ldots, k\}$, X_i is a vertex cut separating s_i from t_i.*

Example 5. Consider a 'local' optimality criterion that, focusing on single conflict t_i, prefers the vertex cut that produces the smallest learned clause, namely, the one minimizing the cardinality $|X_i|$ of the vertex cut X_i. In presence of k conflicts, two natural choices for the 'global' objective function $f(X_1, \ldots, X_k)$, are:

- $f(X_1, \ldots, X_k) = \sum_i |X_i|$, and
- $f(X_1, \ldots, X_k) = |\bigcup_i X_i|$.

In both cases, the weight function $w(\cdot)$ simply evaluates 1 for each node. Clearly, more complex weight function $w(\cdot)$ and objective functions $f(\cdot)$ can be designed. For instance, $w(\cdot)$ might take into account the number of occurrences of literals in the set of clauses, or the *activity* of literals (cf., [15]), to mention two possibilities. □

A further remark is in order. In what follows we will take advantage from the fact that, any given minimum vertex-cut problem can be translated into a minimum edge-cut problem whose solutions correspond to solutions of the former problem. So, in order to simplify the presentation, in the following sections we formalize our learning scheme in terms of (edge) cut and multicut problems.

In the following sections we will describe some alternative approaches to the solution of Problem 1.

4 Solving the multiple learning problem

Let us start by considering the particular case in which the objective function is the sum of the cuts' weights, i.e., $f(X_1, \ldots, X_k) = \sum_i w(X_i)$. In this case, the problem can

[3] In fact, if this is not the case, we can always find k subsets $S_1, \ldots, S_k$ of S, such that S_i contains all sources from which t_i can be reached, and extend G by adding a new vertex s_i having as successors all vertices in S_i (for each i).

[4] For a set of vertices X let $w(X) = \sum_{x \in X} w(x)$.

be formulated as a *minimum s-t multicut problem* [12] (or dually as a *multicommodity maxflow problem* [19]). Alternatively, one can translate the problem into a *vertex separator problem* on a network, as defined, for instance, in [3]. (The reader is referred to the cited literature, and to [2, 7, 28, 29], for a detailed treatment of these problems.)

A drawback of adopting an encoding in an s-t multicut problem is that the obtained solution would consist in a single vertex/edge set X that acts as a separator for each pair s_i-t_i. This does not fulfill the requirement of Problem 1, that asks for a collection of k cuts (each one separating one of the pairs). Consequently, it is necessary to convert the single cut, by splitting it in k subsets (not necessarily pairwise disjoint). In doing this one has to exploit the s_i-t_i-connectedness of G (for each i) in order to determine which part of X is actually related to the pair s_i-t_i. Computing this step may add a computational cost which is in any case polynomially bounded. (Actually, such an overhead could be absent, if connectedness computations are part of the algorithm used to solve the minimum s-t multicut instance).

Recall that, for general graphs, the problem of finding the minimum s-t multicut is Max SNP-hard [8]. Nevertheless, in undirected graphs [12] proposes an $\mathcal{O}(\log k)$ approximate algorithm based on LP relaxation.

An alternative approach consists in solving Problem 1 through the solution of several minimum s-t cut problems. We outline here two possibilities. The first one consists in independently solving k minimum s-t cut problems, one for each pair s_i-t_i (by ignoring, in each of them, all other pairs s_j-t_j, for $j \neq i$). Clearly, the computational cost of this approach depends on the algorithm exploited to solve each of the k simpler problems. For instance, [14] proposes an algorithm for minimum s-t cut, based on push-relabel methods, having $\mathcal{O}(n \times m \times log(n^2/m))$ time complexity. Notice that, the Hao-Orlin algorithm solves this problem by computing $n-1$ s-t cuts and attains an overall complexity which asymptotically matches that of computing a single s-t cut (cf., [6]). Notice that, in general, the union of the k single minimum s-t cuts does not necessarily represent an optimal solution of the s-t multicut problem. Indeed, it can be easily shown that it is a k-factor approximation of such optimal solution. A similar criticism applies with respect to Problem 1, because finding each single s-t cut independently, does not allow one to impose any global optimization criteria.

The second approach can be adopted when the objective function $f(X_1, \ldots, X_k)$ has an intrinsically global nature, i.e., it is not possible to express it as a simple combination of k simpler functions, each concerning a single pair s_i-t_i (as we did by restraining to the minimization of the sum of the cuts' weights), then a more general technique has to be designed. A viable possibility consists in exploiting some kind of enumeration technique (complete or bounded), such as those proposed in [29, 3, 30], in order to compute one sequence of s_i-t_i-cuts, for each i. Then, the solution to Problem 1 can be obtained by suitably combining k single solutions, each one coming from one of the sequences. More details will be provided in the next section.

5 An optimal solution for the general case

In this section we outline a viable algorithmic approach to Problem 1 that guarantees to achieve the optimal solution. The basic idea consists in enumerating, for each of the k

s_i-t_i pairs, the admissible solutions of the corresponding s_i-t_i cut problem. Then, the optimal solution of the global problem can be sought for by evaluating the objective function $f(X_1, \ldots, X_k)$ directly on the k-tuples of single solutions. In this manner one can carry out the optimization of an objective function in its most general form.

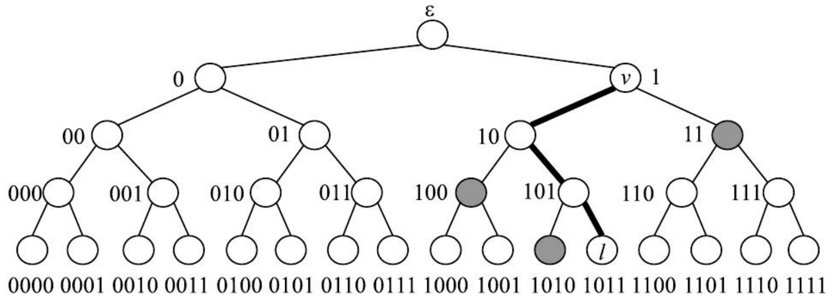
Concerning each single s_i-t_i cut problem, this technique requires to produce an enumeration of all its solutions ordered by non-decreasing weights. Such a specific problem has been formalized precisely in [31] and it turns out to be a $\#P$-complete problem [25].

Concerning the cut problem, instead of the s-t cut problem, a remarkable approach has been proposed in [33], by combining the technique described in [31] with the faster Hao-Orlin algorithm [16]. The resulting algorithm can enumerate all cuts, in non-decreasing order of weight, with a $\mathcal{O}(n \times m \times log(n^2/m))$ delay, between two consecutive solutions.

In our case, we propose the adoption of the same technique (see below), slightly re-engineered in order to obtain an enumeration of all s-t cuts. The resulting naive algorithm is able to solve Problem 1 within $\mathcal{O}(N \times n \times m \times log(n^2/m))$ steps, where N is the number of all s_i-t_i cuts. Notice that, in general, N may be exponential in the size of the given graph [31, 24].

Let us briefly outline the enumeration technique of [33, 31] and how to adapt it to our case. Given a DAG $G(V, E)$, the vertices in V are numbered from 1 to n. In this way, we can represent each cut in G by means of a binary string $b_1, b_2, \ldots, b_n$ composed of n bits, where $b_i = 0$ if and only if the vertex indexed i is in the cut. Conversely, each binary string represents a possible cut.

Consequently, the set of all (strings representing) potential cuts can be organized in a complete binary tree of height n. For example, the picture shown below, borrowed from [33], illustrates such a tree for $n = 4$.



In such a tree, each leaf is associated with one of the possible strings $b_1, b_2, \ldots, b_n$, and *vice versa*. Each internal node, at level h represents a partially specified cut, not involving the vertices indexed from $h + 1$ to n. In other words, such a node of the tree represents the collection of all cuts that agree on the first h vertices. Let this collection be denoted by $C(X, Y)$ with $X = \{i | b_i = 0\}$ and $Y = \{i | b_i = 1\}$. The root of the tree represents all cuts and is denoted by ϵ. For the sake of simplicity, let us identify a node v of the tree with the partially specified cut it represents and let $mc(v)$ denote the minimum cut among those represented by v.

Let $\Pi = \{(\epsilon, mc(\epsilon))\}$ be a working set of pairs where the first component is a node of the tree and the second one is the value of the corresponding minimum cut. The algorithm proceeds by extracting the element $(v, mc(v))$ from Π that minimizes, among the pairs in Π, the value of $mc(v)$. The pair $(v, mc(v))$ is produced as output. Then, the leaf corresponding to the minimum cut $mc(v)$ is considered. Let l be such a cut/leaf (c.f., the above picture). At this point, the path in the tree connecting v to l is considered and for each of the *immediate children* u in such path the pair $(u, mc(u))$ is added to Π. The immediate children are all the nodes which are not in the path, are adjacent to some node in the path, and belong to the tree rooted in v. (They are depicted in gray color in the figure.) Each value $mc(u)$ is obtained by means of an auxiliary computation (for instance, by exploiting the algorithm in [16, 14]). This procedure is repeated until Π is empty.

It is easy to verify that the cuts are enumerated in non-decreasing weight order. Consider once again the above figure. At each step, each of the immediate children u added to Π cannot represent a collection of cuts that includes l. Since l is the optimal solution among all the cuts represented by v, no cut among those represented by u can have weight lower than l. In [31] it is shown that such algorithm, if implemented using suitable data structures, can enumerate all cuts with a $\widetilde{O}(n^2 m)$ time delay.[5]

In order to adapt this technique to our purpose, it suffices to consider that to represent s-t cuts we only need a tree of height $n-2$, because s and t must be separated in each cut. Clearly, an auxiliary algorithm will be used to compute minimum s-t cuts, at each step. It might be the case that not all the leaves of the binary tree actually represent s-t cuts. Hence, there might be internal nodes that represent empty collections of s-t cuts. This particular cases are easily dealt with by simply ignoring the corresponding sub-tree. (Notice that, at least one minimum s-t cut exists and it is determined at the beginning of the execution.)

A very inefficient algorithm for Problem 1 is composed of two steps. First, the above outlined enumeration algorithm is used to compute all s_i-t_i cuts $l_1^{(i)}, \ldots, l_{n_i}^{(i)}$, for each $i \in \{1, \ldots, k\}$, where $n_i \leq 2^{n-2}$ is the number of s_i-t_i cuts. Then, the objective function is optimized on the set of k-tuples of the form $l^{(1)}, \ldots, l^{(k)}$, obtaining the set $\overline{l^{(1)}} \cup \cdots \cup \overline{l^{(k)}}$ that minimizes the value of $f(\overline{l^{(1)}}, \ldots, \overline{l^{(k)}})$.

Clearly, the computational complexity of this algorithm is unsatisfactory, even for small graphs. However, one may exploit the specific properties the implication graph exhibits to gain greater efficiency. The structure of the graph $G(V, E)$ has to be considered. Indeed, G is a layered DAG and the application of suitable heuristics might sensibly improve the naive algorithm. For instance, the Padberg-Rinaldi or the Karger heuristics [6] are certainly exploitable. Moreover, the fact that the DAG is an *implication graph* built up by reflecting the logical relationships between propositional variables, implicitly encoded in the set of clauses, has great relevance. In fact, the DAG encodes logical implications between each vertex/literal and the set of its antecedents. By applying simple propositional properties, one can locally re-write the graph G so as to

[5] Actually, one of the crucial points in reducing the overall complexity of the algorithm consists in handling a set Π of pairs $(v, mc(v))$ instead of simple elements v. In this way, in fact, one has to compute the value of $mc(v)$ just once for each node v (see [31] for the details).

transform it into a graph G' which is equivalent to G w.r.t. the learning process, but belongs to a class of DAGs having better computational properties. Examples of graph classes that are desirable targets for this rewriting process are planar graphs and series-parallel graphs. (Note that instead of explicitly applying these rewritings on the graph, one could encode their effect directly in the algorithms user to compute the cuts.)

Another, not antithetic, possibility consists in introducing a bound on the number of non-decreasing cuts to be computed for each s_i-t_i pair. This, in principle, corresponds to accept a solution that approximates the optimal one. To achieve this, the s-t cut algorithm is used to compute the first $c \leq 2^n$ s_i-t_i cuts, for each i, where c is a fixed constant value; Then, the algorithm proceeds as before, but minimizing the objective function w.r.t. the prefixes of the k sequences of s_i-t_i cuts. The quality of the approximation depends on c and on the specific order in which the single s_i-t_i cuts are enumerated by the s_i-t_i cut algorithm.

Further investigation is needed to study all these issues.

6 Concluding remarks and future work

In this paper we considered one of the most crucial component of modern DPLL-based SAT solvers, namely, the conflict analysis procedure.

The purpose of this procedure is the detection of the reasons behind a failure occurred during the search for a satisfying assignment of a SAT instance. By analyzing the failure, caused by a conflicting set of assignments performed by the solver while visiting a solution space, one or more new clauses are learned and added to the problem being solved. The effect of this addition consists in driving the solver "away from the failure", preventing the solver to make again the same contradictory assignments. Such a kind of solving strategy is typical of the so-called conflict-driven SAT-solvers, but similar techniques have been successfully applied in several other fields of computational logic (such as, ASP, CP, SMT, to mention some).

We proposed a formulation of the conflict analysis task in a form expressive enough to reckon with different orthogonal generalizations of the basic schema. Features such as the analysis of multiple conflicts, the generation of multiple learned clauses, the imposition of global optimality criteria, the treatment of multiple decisions, are easily dealt with in the same framework. Extensions to the case of parallel solvers are also foreseeable.

We formalized the general learning problem as the search for a collection of vertex cuts in a directed acyclic graph. Optimality of the solution is evaluated with respect to a given global objective function. Such a function is basically conceived to express (complex) search strategies and heuristics that control the behavior of the solver. Nevertheless, by considering a single conflict and by choosing a simple objective function (e.g., minimizing set cardinality), one can recover the standard conflict analysis schema, normally exploited in common solvers.

We provided ways to face the general learning problem by exploiting well-known algorithms proposed in literature to solve cut and multicut problems on DAGs.

Clearly, the computational effort required to accomplish the learning task in its most general form, is higher than the one needed to learn a collection of single clauses, each of them justifying a single conflict, independently. The advantage of the general technique comes from the potentially higher quality of the learned (sets of) clauses. For instance, the adoption of suitable global objective functions may enable the identification of common reasons for multiple conflicts or the derivation of conflict clauses that involve "heavy" literals, i.e., relevant literals with respect to the heuristics used to determine vertex weights. This may reduce the number of clauses that are learned. Another possibility consists in encoding in the objective-function criteria that tend to minimize the literals shared among the learned clauses. Consequently, a smaller set of clauses would better affect the search, because they would prune different and distant (i.e., loosely related) portions of the search space.

In case the visit of the search space is split in different searches (this can be achieved by assigning from the beginning a set of variables; this corresponds to making a multiple decision at the first decision level) and/or by running different solvers in parallel, the adoption of a global perspective in conflict analysis may help in learning conflict clauses that act as a communication channel between the different searches.

It should be noted that the difference in the computational efforts required by solving the global problem as a whole and solving k simpler problems, reduces whenever one resorts to approximated algorithms. In this context one might benefit even from recent results on fixed-parameter tractability of multicut problems on DAGs [18]. For example, think about the fact that it seems reasonable to fix the number of conflicts to be considered in each analysis (namely, the parameter k) to a predefined value, or to limit the search to those cuts/clauses having a specific cardinality.

In this paper we restrained ourselves to proposing an initial formalization of a general learning schema. We also outlined some simple algorithmic solutions. Much has to be done and there are many challenging themes for future research. As regards the algorithmic aspects, one may explore the applicability to our context of several heuristics and techniques developed for standard multicut problems, such as, for instance, the Padberg-Rinaldi and the Karger heuristics [6].

Introducing a global perspective for multiple learning is certainly interesting *per se*. It might be the case that one benefits from this new perspective in identifying new learning schemata, different from those currently described in literature, especially concerning parallel solvers.

Clearly, the practical advantage of our proposal has to be validated through an extensive experimental activity. In doing this one may proceed by implementing a concrete prototypical solver. Alternatively, one may integrate a general learning schema into an existing solver. The solvers described in [9, 32, 10], are good candidates for this last possibility. In fact, these solvers (as well as those described in [5, 20, 1, 21], to mention some proposals not necessarily concerned with SAT-solving) take advantage from a high-performance parallel architecture, and offer solid support to the implementation of parallel algorithms for conflict analysis.

Acknowledgments

The authors would like to thank Massimo Bernaschi, Alessandro Dal Palù, Agostino Dovier, Giuseppe Italiano, Luca Labellarte, and Enrico Pontelli for suggestions and enlightening discussions on the topics presented in this paper. This work is supported by the GNCS-2014 project.

References

1. A. Arbelaez and P. Codognet. A GPU implementation of parallel constraint-based local search. In *PDP*, pages 648–655. IEEE, 2014.
2. C. Bentz. On the hardness of finding near-optimal multicuts in directed acyclic graphs. *Theor. Comput. Sci.*, 412(39):5325–5332, 2011.
3. A. Berry, J.-P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. In *Graph-Theoretic Concepts in Computer Science*, pages 167–172. Springer, 1999.
4. A. Biere. *Handbook of satisfiability*, vol. 185. Ios PressInc, 2009.
5. F. Campeotto, A. Dal Palù, A. Dovier, F. Fioretto, and E. Pontelli. Exploring the use of GPUs in constraint solving. In M. Flatt and H.-F. Guo, eds., *PADL*, vol. 8324 of *LNCS*, pages 152–167. Springer, 2014.
6. C. S. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 324–333. Society for Industrial and Applied Mathematics, 1997.
7. M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow: a survey. *European Journal of Operational Research*, 162(1):55–69, 2005.
8. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts. In *Proc. of the 24th ACM Symposium on Theory of Computing*, pages 241–251. ACM, 1992.
9. A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. Exploiting unexploited computing resources for computational logics. In *Proc. of the CILC-12*, vol. 857 of *CEUR Workshop Proceedings*, pages 74–88. CEUR-WS.org, 2012.
10. A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. CUD@SAT: SAT solving on GPUs. *Journal of Experimental and Theoretical Artificial Intelligence*, 2014.
11. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
12. N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.
13. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012.
14. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. of the ACM*, 35(4):921–940, 1988.
15. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.
16. J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proc. of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174. Society for Industrial and Applied Mathematics, 1992.
17. H. Jin and F. Somenzi. Strong conflict analysis for propositional satisfiability. In G. G. E. Gielen, ed., *Proc. of the DATE 2006*, pages 818–823. European Design and Automation Association, Leuven, Belgium, 2006.
18. S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, eds., *Proc. of ICALP*, vol. 7391 of *LNCS*, pages 581–593. Springer, 2012.
19. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. of the ACM*, 46(6):787–832, 1999.
20. P. Manolios and Y. Zhang. Implementing survey propagation on graphics processing units. In A. Biere and C. P. Gomes, eds., *SAT*, vol. 4121 of *LNCS*, pages 311–324. Springer, 2006.
21. R. Marques, L. G. Silva, P. F. Flores, and L. M. Silveira. Improving SAT solver efficiency using a multi-core approach. In C. Boonthum-Denecke and G. M. Youngblood, eds., *FLAIRS Conference*. AAAI Press, 2013.

22. J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
23. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.
24. J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. In V. Rayward-Smith, ed., *Combinatorial Optimization II*, vol. 13 of *Mathematical Programming Studies*, pages 8–16. Springer, 1980.
25. J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.
26. F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.
27. L. Ryan. *Efficient algorithms for clause-learning SAT solvers*. Simon Fraser University, 2004. Master thesis in computer science.
28. H. Shen, K. Li, and S.-Q. Zheng. Separators are as simple as cutsets. In P. S. Thiagarajan and R. H. C. Yap, eds., *Proc. of Advances in Computing Science - ASIAN'99*, vol. 1742 of *LNCS*, pages 347–358. Springer, 1999.
29. H. Shen and W. Liang. Efficient enumeration of all minimal separators in a graph. *Theor. Comput. Sci.*, 180(1-2):169–180, 1997.
30. K. Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discrete Applied Mathematics*, 158(15):1660–1667, 2010.
31. V. V. Vazirani and M. Yannakakis. Suboptimal cuts: Their enumeration, weight and number. In *Automata, Languages and Programming*, pages 366–377. Springer, 1992.
32. F. Vella, A. Dal Palù, A. Dovier, A. Formisano, and E. Pontelli. CUD@ASP: Experimenting with GPGPUs in ASP solving. In *Proc. of the CILC-12*, vol. 1068 of *CEUR Workshop Proceedings*, pages 163–177. CEUR-WS.org, 2013.
33. L.-P. Yeh, B.-F. Wang, and H.-H. Su. Efficient algorithms for the problems of enumerating cuts by non-decreasing weights. *Algorithmica*, 56(3):297–312, 2010.
34. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in Boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.

A metamodeling level transformation from UML sequence diagrams to Coq

Chao Li, Liang Dou and Zongyuan Yang

East China Normal University Shanghai, China
`email: zerochaoli@gmail.com,{ldou,yzyuan}@cs.ecnu.edu.cn`

Abstract. Modeling is an important aspect of UML formal verification that directly affects the quality and efficiency of the verification. Formal models are the foundation of formal verification. As UML diagrams only have semi-formal semantics, they cannot be used for formal verification directly. Recent studies present model transformation from semi-formal UML models to formal models to solve the issues. In this paper, a metamodeling level transformation tool from UML sequence diagrams to formal Coq codes is presented. Using Kermeta (a metamodeling language) and predefined transformation rules that directly added to the metamodel of UML sequence diagrams, models of UML sequence diagrams are transformed into XMI, an intermediate format, and finally to formal Coq codes. This paper is part of our formal verification work for UML sequence diagrams, and the automatically generated Coq codes can be used for further formal verification using the theorem proof assistant Coq in our related works. This paper perfects the whole verification work and provides useful support to improve the integration density of formal verification in the formalization process of UML sequence diagrams.

Keywords: UML sequence diagrams, model transformation, formal verification, metamodeling, Kermeta, Coq.

1 Introduction

Unified Modeling Language (UML) [1] is standardized by Object Management Group (OMG), and has a set of notations to specify and model target system at varying levels of abstraction. For its powerful modelling capability, UML is increasingly popular in the design stage of model-based software development.

Despite the wide use of UML, a number of problems have been identified due to its semi-formal semantics. For example, a developer's understanding of UML models may differ from the designer's understanding, tools for analyzing UML models may be limited to syntactic analysis [2], and system flaws may fail to be revealed in the design phase. In order to provide UML correct foundation of formal semantics, formal methods are getting popular to analyze UML models. Formal methods are the application of precise mathematical fundamentals and techniques to specify systems (formal specification) [3], and provide a systematical way to check the soundness and correctness of system models (formal verification) [4]. Hence, UML formal verification (UFV) makes up for the deficiencies of UML itself and eliminates the inconsistency of different understanding to system design.

UML sequence diagrams have been widely used in the early stage of software development process. Different objects or processes are represented by parallel vertical lines in a sequence diagram. Objects or processes communicate with each other via messages that are represented by horizontal arrows. UML sequence diagrams play an important role in helping developers understand the runtime behaviours of system. Thus, it is important to verify the models when using UML sequence diagrams in the design stage.

A great deal of UFV works have been done, most of them focus on two aspects: the transformation rules from models of UML diagrams to formal models, and the verification process based on the formal models. In previous work [5] [6], we presented formal semantics of UML sequence diagrams and implemented formal verification of the correctness of the semantics in Coq [7], but the transformation for UML sequence diagrams to Coq presentations is implemented manually and transformation rules are not presented systematically. Manual transformation has low efficiency in dealing with large scale models. In this work, we present the systematic transformation rules, and implement the automatic metamodeling level transformation from UML sequence diagrams models to formal Coq presentations, which is the foundation for further formal verification. We have developed a prototype transformation tool using metamodeling languages Kermeta [8].

Metamodeling is the process to define a modeling language completely and precisely. The abstract syntax of modeling language is described by a metamodel, which is also defined in a metamodeling language. A metamodeling language is a superior language to describe modeling languages and it is also defined with a metamodel. The metamodel of a metamodeling language is called metametamodel, which is self-descibing [9]. Model transformation is used to create new models based on existing models. In stead of creating models from scratch, model transformation enables the reuse of information that was once modeled. Metamodeling level transformation ensures that the target model confirms to the target metamodel specification, hence, the transformation is syntactically correct. In addition, metamodeling and model transformation are fully supported by Kermeta. Kermeta is an executable metamodeling language which supports metamodeling level transformation. Moreover, Kermeta stores data of model and metamodels in XML Metadata Interchage (XMI) files, which is widely used among different modeling tools. Hence, it is sufficient for Kermeta to transform UML sequence diagrams to Coq.

The rest of the paper is structured as follows. Firstly, the related work is reviewed in Section 2. Section 3 recalls the model transformation in Kermeta and briefly introduces Coq. Transformation rules and a case study are showed in Section 4. Finally, we conclude in Section 5.

2 Related Work

A variety of formalization work has been proposed for UML diagrams over the years. A formal framework is provided to support visual simulation of UML models that composed of class, object, state, sequence and collaboration diagrams, and an integrated semantics of these models is presented in [11]. However, it only focus on the semantics building and transformation rules of UML diagrams, but further verification of modeling process is not considered. In [12], some useful rules for transforming sequence diagram to petri net are presented, but the transformation process in that work is done manually. In [13], conventional programming language, Java, is used to navigate, create, read or delete models and model elements via specific libraries, all the transformations are at modeling level. However, we use the metamodeling language Kermeta to implement a metamodeling transformation tool, which can transform models of UML sequence diagrams to formal presentations and ensure the syntactic correctness of the transformation at the same time. In [14] [15], UML state diagrams or activity diagrams are firstly formalized with operational semantics, and then translated into input code of formal verification, but they do not provide an automatic transformation tool. In contrast, an automatic translation of state charts and sequence diagrams into generalized stochastic nets is proposed in [16] [17], and their transformation are at metamodeling level.

Our work not only presents transformation rules at metamodeling level, but also implement a transformation process from UML sequence diagrams to Coq codes automatically in Kermeta. The generated codes can be used for further formal verification.

3 Background

3.1 Metamodeling and Model Transformation in Kermeta

Kermeta is an executable metamodeling language which supports describing both structures and behaviours of metamodels. Kermeta is integrated with Eclipse, and distributed as Eclipse plug-in. It is fully compatible with the OMG Essential Meta-Object Facility (EMOF) [18] and Ecore of Eclipse Modeling Framework (EMF) [19]. It provides an action language to specify the body of operations in metamodels. The action language of Kermeta is imperative and object-oriented. It also integrates aspect-oriented features, and supports some design-by-contract features.

As Kermeta relies on EMF for model storage, regular EMF metamodels, Ecore files, can be used. These metamodels can be created and edited using the generic model editor provided with the EMF. Operations can be added to any class in metamodels using the action language provided by Kermeta. In addition, once the source metamodel is created, source model that confirms to the source metamodel can be generated manually using the model editor.

Model transformation in Kermeta takes one source model as input, and produces one target model as output. Both source model and target model should conform to specific metamodel or abstract syntax, and transformation rules should be defined to drive the transformation. That is, given the source model, source and target metamodel (or abstract syntax), and transformation rules, target model can be generated automatically.

In order to write a model transformation in Kermeta, the source and target metamodels (or abstract syntax) should be defined at first. In our work, UML sequence diagrams is the source modeling language and Coq is the target modeling language. Metamodel of UML sequence diagrams and abstract syntax of Coq are explained in the following sections.

3.2 Metamodel of UML Sequence Diagrams

Figure 1 displays the metamodel of UML sequence diagrams which has been defined in Ecore using the EMF editor. This metamodel has been simplified, but covers most of the important elements. `SeqDiagram` represents a model of UML sequence diagrams, it is the top-level class of the metamodel. The main graphical element of the diagram is `Interaction`.

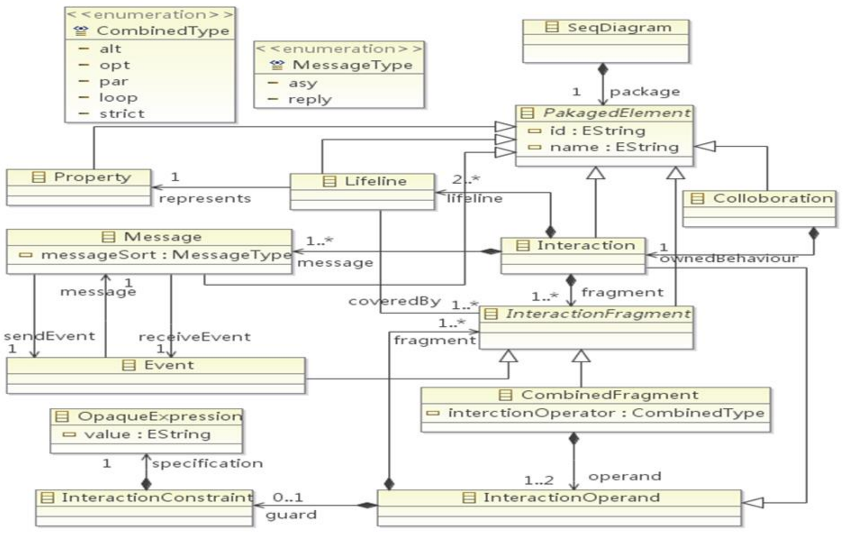


Fig. 1. The Metamodel of UML Sequence Diagram.

`Lifeline`, `Message` and `InteractionFragment` are contained by `Interaction`. Lifeline, message and fragment are the basic elements of UML sequence diagrams. A lifeline represents a specific object. Lifelines communicate with each other through messages, each message triggers two events: send event and receive event. A fragment is an instance of `Event` and `CombinedFragment` that inherit from the abstract class `InteractionFragment`. Fragments describe the behaviour information of UML sequence diagrams. Events are the basic behavioral constructs of UML sequence diagrams and can be combined to form larger behavioral constructs called `CombinedFragment`. A combined fragment consists of an interaction operator, one or more operands which are comprised of events or combined fragments, and an optional guard condition. A combined fragment covers a set of lifeline and decides the execution mode and condition of fragments (events or combined fragments).

3.3 Abstract Syntax of UML Sequence Diagrams in Coq

Coq is a theorem proof assistant. The Calculus of Inductive Constructions (CIC) is the underlying core language of Coq. CIC is based on the calculus of constructions extended by inductive definitions as they are known from the constructive type theory.

The Coq abstract syntax represents UML sequence diagrams as an inductive type as below, which enables reasoning by case analysis and induction.

```
Inductive Seq : Set :=
|Skip : Seq
|E : Event -> Seq
|Alt : Seq -> Seq -> Seq
|Opt : Seq-> Seq
|Strict : Seq -> Seq -> Seq
|Loop : nat -> Seq -> Seq
|Par : Seq -> Seq -> Seq.
```

`Seq` is defined inductively as events and operators in the Coq abstract syntax. `Skip` represents an empty graph. An `E` represents an event. Event is the basic element, it consists of its type and a message, it is defined as :

```
Definition Event := Type * Message.
```

`Type` $\in$ {?,!}, `!` represents sending, and `?` represents receiving. Message is defined as a triple :

```
Definition Message := mName * Lifeline * Lifeline.
```

`mName` represents the name of the message, a message has two lifelines, the first one represents a lifeline sends the message, the second one represents a lifeline receives the message. Furthermore, operators are considered in our work, they decide the execution mode between fragments. We only consider five interaction operators: `alt`, `opt`, `par`, `loop` and `strict`. Among the operators, `opt` is unary and other operators are binary. What calls for special attention is that, two events always occur accompanying a message, these two events are considered as a special combined fragment with strict execution mode.

Models of UML sequence diagrams should be transformed to events and operators definitions that confirm to the abstract syntax. This is discussed in the following section.

4 The metamodeling Transformation work

In the preceding sections, we have described the core concepts of our transformation tool, a more detailed description of the transformation process in our tool is shown in Fig.2.

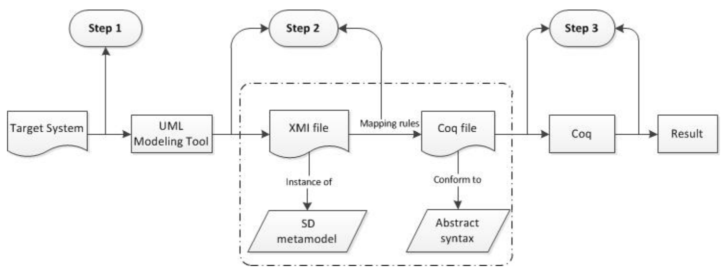


Fig. 2. The outline of our work.

Step1(Manually): Design model of the target system in modeling tools, and load the model into our tool.

Step2(Automatically): Transform the model of UML sequence diagrams to Coq codes with the transformation rules, all these rules are added to classes of UML sequence diagrams metamodel using our tool. This process is also implemented automatically.

Step3(Automatically): Output of model transformation are provided to Coq as input. Further formal verification is based on this output.

4.1 Transformation Rules

Once the source metamodel and the target abstract syntax are defined, models of target system can be transformed to Coq codes confirms to the abstract syntax. The transformation has two steps. In the first step, all the events of source model are transformed to events definition in Coq. In the second step, behaviour elements of source model, i.e. events and combined fragments, are transformed to operators definition in Coq. Before transformation rules of events are given, transformation rule of message is defined in Rule 1.

Rule 1. In UML sequence diagrams, a message is transmitted from one lifeline to another lifeline, this message should be mapped to message variable definition in Coq.

According to Rule 1, we can obtain the Coq codes of the message in Fig.3:

```
Definition m_m1: Message :=("m1","L1","L2").
```

Once the transformation rules of message is defined, Rule 2 for events transformation is given as below.

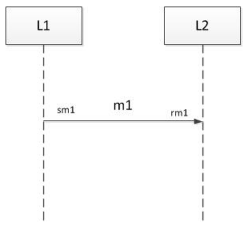


Fig. 3. Message definition.

Rule 2. In UML sequence diagrams, when a message named `m1` is transmitted, two related events occur, one is send event: `sm1`, another is receive event: `rm1`, these two events should be mapped to event variable definition and initialization in Coq.

According to Rule 2, we can obtain the Coq input codes of the events in Fig.3:

```
Definition sm1: Event := (!,m_m1).
Definition rm1: Event := (?,m_m1).
```

According to the two definitions above, we can obtain the ultimate Coq input codes of the events in Fig.3 :

```
Definition sm1: Event := (!,("m1","L1","L2"))
Definition rm1: Event := (?,("m1","L1","L2"))
```

Execution modes between fragments describe the structure information of UML sequence diagrams. In the second step, we define the transformation rules of behaviour information. We start with the transformation rule of events.

Rule 3. In UML sequence diagrams, two events, send event and receive event, always accompanies a message, the execution mode between them is `strict`. Two events of this message are considered as a special combined fragment with `strict` execution mode and should be mapped to operators definition and initialization in Coq.

According to Rule 3, we can obtain the operators codes of the two events in Fig.3:

```
strict(E sm1)(E rm1)
```

Rule 4. In UML sequence diagram, a combined fragment is comprised of one operator and fragments, fragments $\in \{Event, CombinedFragment\}$, and we specify that any two adjacent fragments without operators are in `strict` execution mode. Combined fragments in sequence diagrams should be transformed to operators definition in Coq.

According to the Rule 4, we can obtain the Coq codes of the combined fragment in Fig.4:

```
Alt(Strict (E sm1)(E rm1))(Strict (E sm2)(E rm2))
// sm2 and rm2 is defined in the same way as sm1 and rm1
```

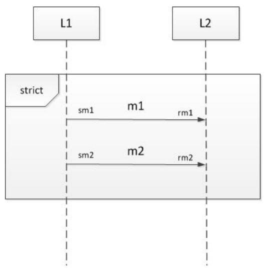


Fig. 4. Combined fragment definition.

4.2 Transformation Algorithm

Using action language and aspect-orientation mechanism in Kermeta, transformation rules can be added to corresponding class in the metamodel of UML sequence diagrams.

```
procedure main()
  //import a sequence diagram model
  seq :SeqModel = loadModel(seq.xmi);
  //call the method toCoq to perform transformation
  coqCode : String = seq.toCoq();
  //save the transformation result
  saveModel(coqCode);
end main
//toCoq is added to the top level class SeqModel
procedure toCoq():String
    result :String;
    foreach m in message
        result = result + m.message2Coq();
    foreach f in fragment
        result = result + f.fragment2Coq();
    return result;
end toCoq
//message2Coq is added to class Message
procedure message2Coq():String
    mName = self.name;
    sLineName = getSendLineName();
    rLineName = getRecLineName();
    sendEvent=write2Coq(!, mName, sLineName, rLineName );
    recEvent = write2Coq(?, mName, sLineName, rLineName );
    return sendEvent + recEvent;
end message2Coq
// fragment2Coq is added to class InteractionFragment
procedure fragment2Coq():String
    //(1)when fragment is event
    if(self.isInstanceOf(OcreenceSpecification))then
        if(self.type ==send)then
        result = result + event2Coq(self.name, send);
    else if(self.type ==receive)then
        result = result + event2Coq(self.name, receive);
    //(2) when fragment is combinedFragment
    else if(self.isInstanceOf(CombinedFragment))then
        if(self.operand == opt)then
            result = result + CombinetoCoq (operand, self.name);
        else
            leftOp = self;
            rightOp = nextFragment;
            result = result + CombinetoCoq(operand, leftOp, rightOp);
        //transform every subfragment in combinedfragment
        foreach f in self.operand.fragment
           result = result + f.fragment2Coq();
    return result;
end fragment2Coq
```

Operation `event2Coq` transforms a send or receive event to Coq codes,and operation `Combine2Coq` transforms a combined fragment with unary or binary operators to Coq codes.

4.3 A Case Study

In this section, a simple example is presented to illustrate our transformation. Fig.5 shows a scene that a user sends his account *id* and *password* to the Automatic Teller Machine (ATM)and get a reply from it. If logged in successfully, the user can check balance or withdraw money.

After reading the XMI file of Fig.5 and parse it, our transformation tool automatically extract the useful information and transform it to formal Coq codes that confirms to the abstract syntax we have defined:

```
Definition sid :Event :=(!,("id","User","ATM")
Definition rid :Event :=(?,("id","User","ATM")
Definition spwd :Event :=(!,("pwd","User","ATM")
Definition rpwd :Event :=(?,("pwd","User","ATM")
Definition sloginSucc :Event :=(!,("loginSucc","ATM","User")
Definition rloginSucc :Event :=(?,("loginSucc","ATM","User")
Definition swithdraw :Event :=(!,("withdraw","User","ATM")
Definition rwithdraw :Event :=(?,("withdraw","User","ATM")
Definition scheck :Event :=(!,("check","User","ATM")
Definition rcheck :Event :=(?,("check","User","ATM")
Definition sloginFail :Event :=(!,("loginFail","ATM","User")
Definition rloginFail :Event :=(?,("loginFail","ATM","User")
Definition ExampleSeq :Seq :=
```

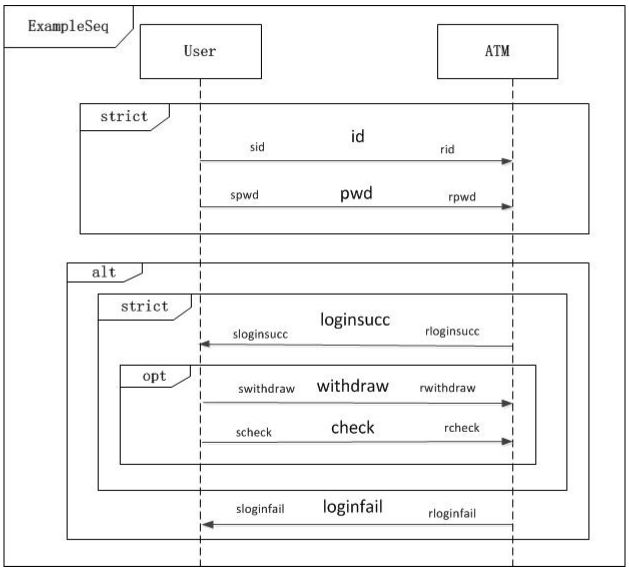


Fig. 5. An example model of UML Sequence Diagram.

```
Strict (Strict (Strict (E sid)(E rid))(Strict (E spwd)(E rpwd)))
(Alt(Strict(Strict (E sloginsucc)(E rloginsucc))(Opt (Strict (Strict
(E swithdraw)(E rwithdraw))(Strict (E scheck)(E rcheck)))))
(Strict (E sloginfail)(E rloginfail))).
```

5 Conclusion

Modeling is an important step in the UML diagrams formalization, it lays the foundation for further formal verification. In this paper, we focus on the model transformation for UML sequence diagrams and implement a metamodeling transformation tool in Kermeta. Firstly, metamodel of UML squence diagrams and exact definition of Coq abstract syntax are given. Then, using predefined transformation rules that directly added to the classes in UML sequence diagrams metamodel, models of UML sequence diagrams are automatically transformed to Coq codes so that further verification could be done. Finally, we present a case study to show the result of model transformation. The result can be as the input codes for formal verification in Coq theorem prover.

We consider model concepts of sequence diagrams but not the whole, we can further define and extend our transformation rules. Another future direction is to extend the transformation to implement the transformation of UML state diagrams. In addition, our transformation tool is integrated with Kermeta, but separated with the verification process, we hope to combine them together and package them as a new exe or web-based tool in future.

Acknowledgment

This work is supported by National Natural Science Foundation of China (No.61070226).

References

1. J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*, 1999.
2. R. France, *The UML as a formal modeling notation*, Computer Standards and Interfaces, vol. 19, pp. 325–334, 1998.
3. M. Gogolla, F. Bttner, and M. Richters, *USE: A UML-based specification environment for validating UML and OCL*, Science of Computer Programming, vol. 69, pp. 27–34, 2007.
4. R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers, *Using description logic to maintain consistency between UML models*, Modeling Languages and Applications, pp. 326–340, 2003.
5. Y. Zuo, L. Dou, L. Xu, and Z. Yang, *Mechanized sementics of UML sequence diagrams*, International Conference on Engineering and Applied Science, Colombo, Sri Lanka, Dec. 27-29 2012.
6. L. Dou, L. Lu, Z. Yang, and J. Xie, *Towards mechanized semantics of UML sequence diagrams and refinement relation*, The 24th IASTED International Conference on Modelling and Simulation, Banff, Canada, vol. 69, July 17-19 2013.
7. Coq, http://coq.inria.fr.
8. Kermeta, http://www.kermeta.org.
9. D. Cetinkaya and A. Verbraeck, *Metamodeling and model transformations in modeling and simulation*, Proceedings of the Winter Simulation Conference, Winter Simulation Conference, 2011.
10. OMG, *XML Metadata Interchange, version 1.2*, http://www.omg.org/, 2002.
11. M. Gogolla, P. Ziemann, and S. Kuske, *Towards an integrated graph based semantics for uml*, Electr. Notes Theor. Comput. Sci, vol. 72, 2003.
12. O. R. Ribeiro and J. M. Fernandes, *Some rules to transform sequence diagrams into Coloured Petri Nets*,7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006), pp. 37–56, 2006.
13. D. H. Akehurst, B. Bordbar, and M. J. Evans, *SiTra: Simple transformations in java*, Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, pp. 351–364.
14. S. Gnesi and F. Mazzanti, *A model checking verification environment for UML statecharts*, In XLIII Annual Italian Conference AICA, Udine, 2004.
15. R. Eshuis, *Symbolic model checking of UML activity diagrams*, ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 15, pp. 1–38, 2006.
16. S. Bernardi, S. Donatelli, and J. Merseguer, *From UML sequence diagrams and statecharts to analysable Petri Net models*, Proceedings of the 3rd international workshop on Software and performance, 2002.
17. S. Bernardi and J. Merseguer, *Performance evaluation of UML design with stochastic well-formed nets*, Journal of Systems and Software, vol. 11, pp. 1843–1865, 2007.
18. OMG, *MOF 2.0 Core Final Adopted Specification*, http://www.omg.org/cgi-bin/doc?ptc/03-10-04, 2004.
19. Mark A. Pinsky, *The EMF Book*, Warner Books,1995.

An efficient algorithm for generating symmetric ice piles*

Roberto Mantaci[1], Paolo Massazza[2], and Jean-Baptiste Yunès[1]

[1] LIAFA, Université Paris 7 Denis Diderot, Case 7014, 75205 Paris Cedex 13, France
Roberto.Mantaci@liafa.univ-paris-diderot.fr,
Jean-Baptiste.Yunes@univ-paris-diderot.fr

[2] Università degli Studi dell'Insubria, Dipartimento di Scienze Teoriche e Applicate - Sezione Informatica, Via Mazzini 5, 21100 Varese, Italy
paolo.massazza@uninsubria.it

Abstract. We define the *Symmetric Ice Pile Model* $\mathrm{SIPM}_k(n)$, a generalization of the *Ice Pile Model* $\mathrm{IPM}_k(n)$, and we show an efficient algorithm for generating the symmetric ice piles with n grains. More precisely, we show how to exploit an existing algorithm for generating $\mathrm{IPM}_k(n)$ in order to generate $\mathrm{SIPM}_k(n)$ in amortized time $O(1)$ and in space $O(\sqrt{kn})$.

1 Introduction

In this paper, we consider the problem of generating particular unimodal sequences that describe the reachable states of the symmetric version of the well-known *Ice Pile Model* $\mathrm{IPM}_k(n)$, a discrete dynamical system introduced by Goles Morvan and Phan [7] as a restriction of the discrete dynamical model proposed in 1973 by Brylawski [2] in order to study linear partitions of an integer n.

$\mathrm{IPM}_k(n)$ admits a description in terms of a simple game that simulates the movements of ice grains organized into adjacent columns of decreasing heights. If there are two adjacent columns, say i and $i+1$, with heights differing by at least 2, a grain can fall down from column i to column $i+1$ (the game starts with n grains stacked in column 0). Moreover, if a column i of height p is separated from a column j of height $p-2$ by a plateau of at most $k-1$ columns of height $p-1$, then a grain can slide from i to j crossing the plateau. $\mathrm{IPM}_k(n)$ is an extension of the *Sand Pile Model* $\mathrm{SPM}(n)$, in which only the fall rule is permitted. $\mathrm{SPM}(n)$ has been widely studied in combinatorics, poset theory, physics, and in the theory of cellular automata to represent granular objects, see [1], [6], [7], [8].

A natural extension of $\mathrm{SPM}(n)$ has been introduced [5], [12] in order to fix the lack of symmetry (grains either stay or move to the right). Thus, a grain possibly falls down nondeterministically from column i either to column $i-1$ or to column $i+1$. This new model, called *Symmetric Sand Pile Model* and denoted by $\mathrm{SSPM}(n)$, consists of all the integer sequences describing the reachable states

* Partially supported by Project M.I.U.R. PRIN 2010-2011: Automi e linguaggi formali: aspetti matematici e applicativi

of a system starting with n grains in column 0 (the index of a column can be negative).

Despite the simplicity of this rule, the underlying structure drastically changes. While SPM(n) turns out to be a distributive lattice with exactly one fixed point (the bottom, easily characterized), SSPM(n) is neither a lattice nor admits a unique fixed point. Nevertheless, a characterization of fixed points exists [5, Section 3.2], [12, Th. 2] and their number is known [5, Lemmas 15,16,17].

In this paper we introduce the symmetric version of $\mathrm{IPM}_k(n)$, denoted by $\mathrm{SIPM}_k(n)$, where left and right slide moves on plateaux of length smaller than or equal to $k-1$ are also permitted, in addition to the left and right fall rules. In particular, we show that $\mathrm{SIPM}_k(n)$ can be generated by means of a CAT (Constant Amortized Time) algorithm. We recall that CAT algorithms for generating sand and ice piles have been presented in [9] and [10], and that a CAT algorithm for generating symmetric sand piles has been proposed in [11], where a decomposition property of symmetric sand piles in terms of product of sand piles is exploited. We prove that a similar decomposition property holds for symmetric ice piles: this lets us design a CAT algorithm that sequentially generates the elements of $\mathrm{SIPM}_k(n)$ using $O(\sqrt{kn})$ space.
In Section 2 we lay down preliminaries such as definitions as well as properties and characteristics of unimodal sequences and generalized unimodal sequences, the combinatorial objects used for modeling symmetric ice piles. We also recall some properties of sand and ice piles, as well as the basics of the CAT algorithm used to generate $\mathrm{IPM}_k(n)$. In Section 3 we characterize unimodal sequences that can be forms of symmetric ice piles and generalized unimodal sequences that can be configurations of $\mathrm{SIPM}_k(n)$. These properties are the key for proving that our algorithm is correct and computing its complexity. The algorithm is outlined in Section 4 whereas its complexity is analysed in Section 5.

2 Preliminaries

A linear partition of n is a non-increasing sequence of strictly positive integers, $s = (s_0, \dots, s_l)$, such that $\sum_{i=0}^{l} s_i = n$; the *height*, the *length* and the *weight* of s are $h(s) = s_0$, $l(s) = l+1$ and $w(s) = n$, respectively. We consider the set LP(n) of linear partitions of n equipped with the negative lexicographic or *neglex* ordering, $<_{\text{nlex}}$, defined as follows: $(s_0, \dots, s_l) <_{\text{nlex}} (t_0, \dots, t_m)$ if and only if there exists i, $0 \le i \le \min(l, m)$ such that $s_i > t_i$ and $s_j = t_j$ for $j < i$. A *unimodal sequence* of n is a sequence of strictly positive integers, $a = (a_0, \dots, a_l)$, such that $\sum_{i=0}^{l} a_i = n$ and $a_0 \le a_1 \le \dots \le a_j \ge a_{j+i} \ge \dots \ge a_l$ for some j. The smallest index q such that $a_{q-1} \le a_q \ge a_{q+1} \ge \dots \ge a_l$ is called the *center* of a, noted by $c(a)$. Obviously, $h(a) = a_{c(a)}$ and, if a is a constant sequence then $c(a) = 0$. From here on, for $i \ge 0$ we let $a_{<i} = (a_0, a_1 \dots, a_{i-1})$ and $a_{\ge i} = (a_i, a_{i+1}, \dots, a_l)$. Given $h > 1$, we indicate by $p^{[h]}$ the sequence $\underbrace{(p, p, \dots, p)}_{h}$, interpreted as a *plateau* of h columns of height p. The *catenation* product of sequences is denoted by "$\cdot$", $(a_0, \dots, a_l) \cdot (b_0, \dots, b_p) = (a_0, \dots, a_l, b_0, \dots, b_p)$, and the reversal of a by $\overline{a} = (a_l, \dots, a_0)$. If $a = b \cdot c$ we say that b and c are a *prefix* and a *suffix* of a, respectively.

We equip the set $\mathrm{US}(n)$ of unimodal sequences of n with a particular linear order " $<$ ". Thus, let $a, b \in \mathrm{US}(n)$, $x_1 = h(a)$, $x_2 = h(b)$ and consider the (unique) factorizations $a = c \cdot x_1^{[p_1]} \cdot d$, $\quad b = e \cdot x_2^{[p_2]} \cdot f$ with $h(c), h(d) < x_1$ and $h(e), h(f) < x_2$. We say that $a < b$ if and only if either $(x_1 > x_2)$ or $(x_1 = x_2, p_1 > p_2)$ or $(x_1 = x_2, p_1 = p_2, w(d) > w(f))$ or $(x_1 = x_2, p_1 = p_2, w(d) = w(f), d <_{\mathrm{nlex}} f)$ or $(x_1 = x_2, p_1 = p_2, d = f, c <_{\mathrm{nlex}} e)$.

By possibly considering negative indices, we say that $(a_j, a_{j+1} \dots, a_l)$ is a *generalized unimodal sequence* of *form* $b = (b_0, \dots, b_{l-j})$ if and only if b is a unimodal sequence and $b_i = a_{i+j}$, $0 \le i \le l - j$. Index j is the *position* of a. In other words, a generalized unimodal sequence is obtained by a unimodal sequence by (right- or left-) shifting all its entries by j places. We identify a generalized unimodal sequence by means of a pair (a, j) where a is a unimodal sequence (the form) and j an integer (the position). We also write $\boldsymbol{a}$ to indicate a generalized unimodal sequence whenever the value j does not matter.

We extend the linear order $<$ to the set $\mathrm{GUS}(n)$ of generalized unimodal sequences of n by setting $(a, i) < (b, j)$ if and only if either $a < b$ or $a = b$ and $i < j$. Trivially, one has $\mathrm{LP}(n) \subset \mathrm{US}(n) \subset \mathrm{GUS}(n)$. If $\boldsymbol{a} = (a, j) \in \mathrm{GUS}(n)$ then for any i with $j \le i \le j + l(a) - 1$ we set $\boldsymbol{a}_{<i} = (a_j, \dots, a_{i-1})$ and $\boldsymbol{a}_{\ge i} = (a_i, \dots, a_{j+1(a)-1})$. The *right height difference* of $\boldsymbol{a} = (a, j)$ at i is defined as $\delta_r(\boldsymbol{a}, i) = a_i - a_{i+1}$ (assume $a_i = 0$ for $i > j + l(a) - 1$ or $i < j$). Analogously, the *left height difference* of $\boldsymbol{a}$ at i is $\delta_l(\boldsymbol{a}, i) = a_i - a_{i-1}$.

We interpret the elements of $\mathrm{GUS}(n)$ as blocks of n grains disposed into adjacent columns and define four (partial) functions called *moves*. Let $\boldsymbol{a} = (a, j) \in \mathrm{GUS}(n)$, then the right fall of a grain in column i with $j \le i \le j + l(a) - 1$ is

$$\mathrm{RFall}(\boldsymbol{a}, i) = \begin{cases} (a_j, \dots, a_{i-1}, a_i - 1, a_{i+1} + 1, \dots, a_{j+l(a)-1}) & \text{if } \delta_r(\boldsymbol{a}, i) > 1, \\ \perp & \text{otherwise} \end{cases}$$

The function RSlide_k allows the crossing of a plateau of length at most $k - 1$:

$$\mathrm{RSlide}_k(\boldsymbol{a}, i) = (a_j, \dots, a_{i-1}, \underbrace{p, p, \dots, p}_{k'+2}, a_{i+k'+2}, \dots, a_{j+l(a)-1})$$

if there is $k' < k$ such that

$$\boldsymbol{a} = (a_j, \dots, a_{i-1}, p + 1, \underbrace{p, p, \dots, p}_{k'}, p - 1, a_{i+k'+2}, \dots, a_{j+l(a)-1})$$

otherwise $\mathrm{RSlide}_k(\boldsymbol{a}, i) = \perp$. The functions $\mathrm{LFall}(\boldsymbol{a}, i)$, and $\mathrm{LSlide}_k(\boldsymbol{a}, i)$ are defined symmetrically.

Whenever the value k is obvious, we simply write $\boldsymbol{a} \overset{i}{\Rightarrow} \boldsymbol{b}$ if either $\boldsymbol{b} = \mathrm{LFall}(\boldsymbol{a}, i)$ or $\boldsymbol{b} = \mathrm{RFall}(\boldsymbol{a}, i)$ or $\boldsymbol{b} = \mathrm{LSlide}_k(\boldsymbol{a}, i)$ or $\boldsymbol{b} = \mathrm{RSlide}_k(\boldsymbol{a}, i)$. In general, we write $\boldsymbol{a} \overset{\star}{\Rightarrow} \boldsymbol{b}$ if there is a sequence of moves leading from $\boldsymbol{a}$ to $\boldsymbol{b}$.

2.1 The Ice Pile Model

$\mathrm{IPM}_k(n)$ consists of the closure of $\{(n)\}$ under RFall and RSlide_k. Linear partitions of $\mathrm{IPM}_k(n)$ have been characterized combinatorially in [7, Th. 3].

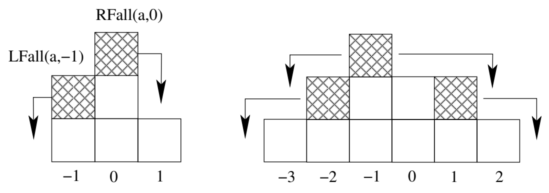


Fig. 1. Moves in GUS(n): RFall and LFall, RSlide$_3$ and LSlide$_3$.

Such theorem implies two upper bounds that are important for our purposes :

- For any ice pile a with height h one has $w(a) \leq h + k\frac{(h+1)h}{2}$.
- For any ice pile $a \in \mathrm{IPM}_k(n)$ one has $l(a) \leq O(\sqrt{kn})$.

Given $a, b \in \mathrm{IPM}_k(n)$ we say that b *dominates* a, noted by $a \prec b$, if and only if for all $e \geq 0$ one has $\sum_{i=0}^{e} b_i \geq \sum_{i=0}^{e} a_i$. Note that $a \prec b$ implies $b <_{\mathrm{nlex}} a$. The *dominance* order $\prec$ is related to the order induced by $\Rightarrow$. Indeed, if $a \prec b$ then $b \overset{\star}{\Rightarrow} a$ (see [7]).

An ice pile a is called a *staircase of height* h if and only if either $a = h^{[k_0]} \cdot (h - j_0)$, with $0 < k_0 < k$ and $0 \leq j_0 \leq h$, or

$$a = h^{[k]} \cdot (h-1)^{[k]} \cdot (h-2)^{[k]} \cdots (h-i)^{[k]} \cdot (h-i-1)^{[k_1]} \cdot (h-i-1-j)$$

with $i \geq 0$, $0 \leq k_1 < k$ and $j > 0$. We indicate by $\mathrm{IPM}_{k,h}(n)$ the set of ice piles of height at most h with n grains, $\mathrm{IPM}_{k,h}(n) = \{a \in \mathrm{IPM}_k(n) \mid h(a) \leq h\}$. Note that for a suitable n with $(k+1)h < n \leq h + k\frac{(h+1)h}{2}$, the ice pile $\min_{<_{\mathrm{nlex}}}(\mathrm{IPM}_{k,h}(n))$ is

$$h^{[k+1]} \cdot (h-1)^{[k]} \cdot (h-2)^{[k]} \cdots (h-i)^{[k]} \cdot (h-i-1)^{[k_1]} \cdot (h-i-1-j).$$

Definition 1. *For any $h > 0$, we call h-*critical *an ice pile $a \in IPM_{k,h-1}(n))$ such that $h^{[k+1]} \cdot a$ is not an ice pile.*

An ice pile $a \in \mathrm{IPM}_{k,h}(n))$ is $(h+1)$-*critical* if and only if it has a prefix in the set of ice piles

$$\Pi_h = \{a \in \mathrm{IPM}_k(r) | r > 0 \wedge \exists e, 0 \leq e < h, a = \prod_{i=0}^{e} (h-i)^{[k]}(h-e)\}.$$

The *set of moves* of a is $\mathrm{M}(a) = \{i | 0 \leq i \leq l(a),\ \mathrm{RFall}(a,i) \neq \perp \text{ or } \mathrm{RSlide}_k(a,i) \neq \perp\}$. If a is a staircase or $a = \min_{<_{\mathrm{nlex}}}(\mathrm{IPM}_{k,h}(n))$ then $|\mathrm{M}(a)| \leq 3$, indeed $\mathrm{M}(a) \subseteq \{l(a) - j, l(a) - 2, l(a) - 1\}$ for a suitable $j \leq k$. If $\mathrm{M}(a) = \emptyset$ then a is a *fixed point.* We denote by $a^{(e)}$ the eth ice pile of $\mathrm{IPM}_k(n)$ (w.r.t. $<_{\mathrm{nlex}}$) and by $G(a)$ the set of ice piles generated from a, $G(a) = \{b | a \overset{\star}{\Rightarrow} b\}$.

Lemma 1. *Let $a = \min_{<_{nlex}}(IPM_{k,h}(n))$. Then $G(a) = IPM_{k,h}(n)$.*

All ice piles of height h which are smaller than a staircase of height h are $(h+1)$-critical:

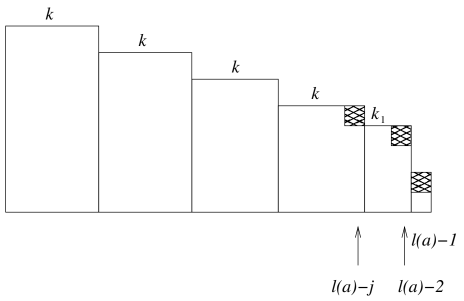


Fig. 2. Possible moves for a staircase.

Lemma 2. *Let $a, b \in IPM_{k,h}(n)$ and suppose that a is a staircase of height h. Then $(b <_{nlex} a) \iff b$ is $(h+1)$-critical.*

Corollary 1. *Let $a \in IPM_{k,h}(n)$ be a staircase of height h. Then,*

$$G(a) = \{b \in IPM_{k,h}(n) | b \text{ is not } (h+1)\text{-critical}\}.$$

For complexity evaluation purposes, we give a bound for the number of ice piles generated by a staircase or by the smallest ice pile of given height. (see fig. 3).

Lemma 3. *Let a be either $\min_{<_{nlex}}(IPM_{k,h}(n))$ or a staircase in $IPM_{k,h}(n)$. If a is not a fixed point then $|G(a)| = \Omega(\frac{l(a)}{k})$.*

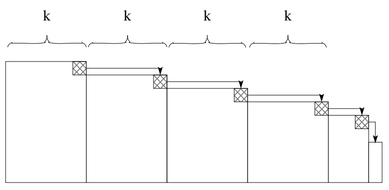


Fig. 3. A sequence of $\Omega(\frac{l(a)}{k})$ moves in a staircase.

We recall here a result [10, Lemma 2.16] on which the CAT generation of $\mathrm{IPM}_k(n)$ is based.

Lemma 4. *For any $e > 0$, let $i_e = \max(M(a^{(e)}))$. Then we have*

$$A(a^{(e)}) \overset{i_e}{\Rightarrow} a^{(e+1)}$$

where $A(a^{(e)}) = \min_{<_{nlex}}\{a \in IPM_k(n) \mid a_{<i_e+1} = a^{(e)}_{<i_e+1}, i_e \in M(a)\}$.

We consider a function Next : $\mathrm{IPM}_k(n) \mapsto \mathrm{IPM}_k(n)$ such that for $e > 0$, $\mathrm{Next}(a^{(e)}) = a^{(e+1)}$ ($\mathrm{Next}(a^{(e)}) = \perp$ if $a^{(e)}$ is a fixed point). In [10] it is shown how this function can be implemented so that if $a = (n)$ then, for any fixed integer k, the iteration of the instruction $a{:=}\textsc{Next}(a)$ (until $\mathrm{M}(a) \neq \emptyset$) generates $\mathrm{IPM}_k(n) = G((n))$ in time $O(|G((n))|)$ (and so is a CAT algorithm) using $O(\sqrt{n})$ space. More generally, for any $a \in \mathrm{IPM}_k(n)$ one can generate $G(a)$ in time $O(|G(a)|)$.

3 The Symmetric Ice Pile Model

In [5] and [12] the Symmetric Sand Pile Model SSPM(n) (obtained by closing $\{(n)\}$ with respect to RFall and LFall) has been studied and its relation with SPM(n) analysed. Here we define the *Symmetric Ice Pile Model* $\mathrm{SIPM}_k(n)$ as the set of integer sequences (called symmetric ice piles) obtained by closing $\{(n)\}$ w. r. t. RFall, RSlide_k, LFall and LSlide_k. Note that if $(a, j) = (a_j, \ldots, a_{l(a)+j-1}) \in \mathrm{GUS}(n)$ is reached from the initial configuration (n) (all the grains in column 0) then one has $j \leq 0 \leq l(a) + j - 1$, since the evolution rules never empty a column completely (positions j of sequences in $\mathrm{SIPM}_k(n)$ are nonpositive).

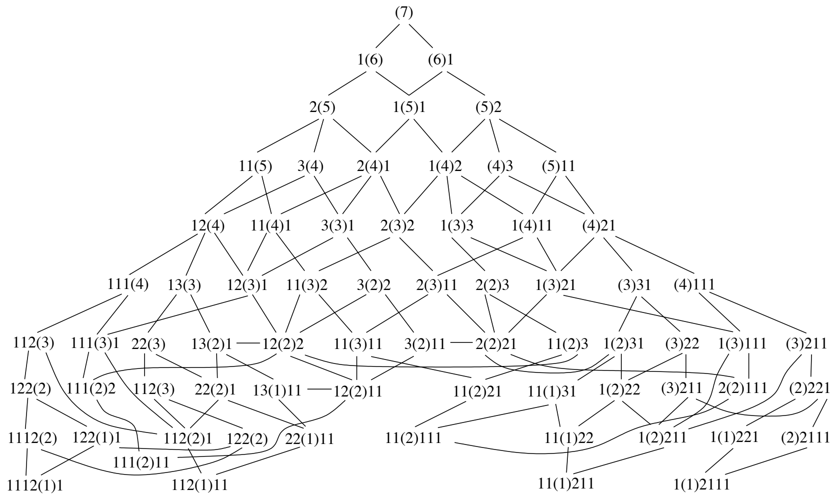


Fig. 4. The poset $\mathrm{SIPM}_2(7)$ (w.r.t. the order relation induced by $\Rightarrow$).

Figure 4 shows several elements having the same form but with different positions (parentheses are used to indicate column 0). For example $(111211, -2)$, $(111211, -3)$ and $(111211, -4)$ are all in $\mathrm{SIPM}_2(7)$.

As a matter of fact, symmetric ice piles are closely related to ice piles.

Definition 2. *Let $a = (a_0, \ldots, a_l)$ be a unimodal sequence. Then a is decomposable at i if and only if $a_i = h(a)$ and $\overline{a_{<i}}, a_{\geq i}$ are ice piles.*

The elements of $\mathrm{SIPM}_k(n)$ have a form which can be decomposed into the concatenation of a reversed ice pile with an ice pile. More precisely, we extend to $\mathrm{SIPM}_k(n)$ a decomposition property proved for $\mathrm{SSPM}(n)$ in [12, Lemma 3].

Lemma 5. *Let $a \in US(n)$, then the two following conditions are equivalent:*

1. *there is an integer i such that a is decomposable at i, with $\overline{a_{<i}} \in IPM_k(r)$, $a_{\geq i} \in IPM_k(s)$ and $r + s = n$;*
2. *there is an integer j such that $(a, j) \in SIPM_k(n)$.*

Definition 3. *For $a \in US(n)$ (resp. $\boldsymbol{a} \in GUS(n)$) we denote by $D(a)$ (resp. $D(\boldsymbol{a})$) the set of all indices i such that a (resp. $\boldsymbol{a}$) admits a decomposition at i (or equivalently, is decomposable at i).*

Remark 1. Obviously, if $\boldsymbol{a} = (a, j) \in \mathrm{GUS}(n)$, one has $D(\boldsymbol{a}) = D(a) + j = \{i + j \mid i \in D(a)\}$. Note also that $D(\boldsymbol{a})$ is always an integer interval. Indeed, the set $\{e \mid \boldsymbol{a}_e = h(\boldsymbol{a})\}$ is an integer interval $[e_1, e_2]$ (with $e_1 = c(\boldsymbol{a})$) and then $D(\boldsymbol{a}) = [i_1, i_2]$ where :

$$i_1 = \begin{cases} max(e_1, e_2 - k + 1), & \text{if } \boldsymbol{a}_{\geq e_2+1} \text{ is } h(\boldsymbol{a})\text{-critical} \\ max(e_1, e_2 - (k+1) + 1) = max(e_1, e_2 - k), & \text{otherwise} \end{cases}$$

$$i_2 = \begin{cases} min(e_2, e_1 + k - 1), & \text{if } \overline{\boldsymbol{a}_{<e_1}} \text{ is } h(\boldsymbol{a})\text{-critical} \\ min(e_2, e_1 + (k+1) - 1) = min(e_2, e_1 + k), & \text{otherwise} \end{cases}$$

Now, we want to determine when a *generalized* unimodal sequence belongs to $\mathrm{SIPM}_k(n)$. Following [12], we start defining a unilateral (possibly reversed) ice pile $f(\boldsymbol{a}, i)$ associated with a pair $\boldsymbol{a} \in \mathrm{GUS}(n)$ and $i \in D(\boldsymbol{a})$.

Definition 4. *Let $\boldsymbol{a} \in GUS(n)$ and $i \in D(\boldsymbol{a})$. If $i \geq 0$, we define $f(\boldsymbol{a}, i)$, called the* completion of $\boldsymbol{a}$ at i, *as the ice pile $s = (s_0, s_1, \ldots)$* of minimal weight *such that $s_{\geq i} = \boldsymbol{a}_{\geq i}$. In this case we will call* complement of $\boldsymbol{a}$ at i *the ice pile $c(\boldsymbol{a}, i) = (s_0, s_1, \ldots, s_{i-1})$.*
Similarly, if $i < 0$, we define $f(\boldsymbol{a}, i)$ as the generalized unimodal sequence $\boldsymbol{s} = (\ldots, s_{-2}, s_{-1}, s_0)$ such that s is the reversed ice pile of minimal weight with $\boldsymbol{s}_{<i} = \boldsymbol{a}_{<i}$. In this case the complement of $\boldsymbol{a}$ at i *is the reversed ice pile $c(\boldsymbol{a}, i) = (s_i, s_{i+1}, \ldots, s_{-1}, s_0)$. In either case, $w(\boldsymbol{a}, i)$ denotes the weight of $f(\boldsymbol{a}, i)$.*

Example 1. Let $n = 21$, $k = 2$, $a = (1, 2, 2, 3, 4, 4, 4, 1)$ and $\boldsymbol{a} = (a, -1)$. Then $c(\boldsymbol{a}, 3) = (6, 5, 5)$ and hence $f(\boldsymbol{a}, 3) = (6, 5, 5, 4, 4, 4, 1)$, while $c(\boldsymbol{a}, 5) = (6, 5, 5, 4, 4)$ but still the same completion $f(\boldsymbol{a}, 5) = (6, 5, 5, 4, 4, 4, 1) = f(\boldsymbol{a}, 3)$. To study a case with $i < 0$, let $b = (1, 2, 2, 3, 4, 4, 4, 1)$ and $\boldsymbol{b} = (b, -7)$. Then $f(\boldsymbol{b}, -2) = ((1, 2, 2, 3, 4, 4, 4, 5), -7) = f(\boldsymbol{b}, -1) = f(\boldsymbol{b}, -3)$.

For any symmetric ice pile $\boldsymbol{a}$ which is decomposable at i, the weight of $c(\boldsymbol{a}, i)$ is uniquely determined by either $\boldsymbol{a}_{<i}$ $(i < 0)$ or $\boldsymbol{a}_{\geq i}$ $(i \geq 0)$ and easily computed.

Lemma 6. *Let $\boldsymbol{a} = (a, j) \in GUS(n)$ be decomposable at i. Then $w(c(\boldsymbol{a}, i))$ can be computed in time $O(1)$.*

The following Lemma is used for proving Lemma 8, which provides a characterization for the configurations of $\mathrm{SIPM}_k(n)$.

Lemma 7. *Let $\boldsymbol{t} \in SIPM_k(n)$ such that $\boldsymbol{t} = (t, 0)$ and let $\boldsymbol{a} = (a, j)$ be another generalized unimodal sequence such that $j < 0$, $l(a) = |j| + l(t)$, $\overline{a} \in IPM_k(n)$ and such that for all i with $0 \leq i < l(t)$ one has $a_i \leq t_i$. Then $\boldsymbol{a}$ is reachable from $\boldsymbol{t}$.*

Lemma 8. *A sequence $\boldsymbol{a} = (a, j) \in GUS(n)$ belongs to $SIPM_k(n)$ if and only if $\exists\ i$ with $j \leq i \leq l(a) - 1 + j$ such that $\boldsymbol{a}$ is decomposable at i and $w(\boldsymbol{a}, i) \leq n$.*

The condition of the previous lemma characterizing generalized unimodal sequences belonging to $\mathrm{SIPM}_k(n)$, can be replaced by an equivalent (although apparently stronger) condition.

Lemma 9. *Let $\boldsymbol{a} = (a, j) \in GUS(n)$. Then, there exists an integer $i_0 \in D(\boldsymbol{a})$ such that $w(\boldsymbol{a}, i_0) \leq n$ if and only if for all $i \in D(\boldsymbol{a})$, one has $w(\boldsymbol{a}, i) \leq n$.*

Corollary 2. *Let $\boldsymbol{a} \in GUS(n)$. If $0 \in D(\boldsymbol{a})$ then $\boldsymbol{a} \in SIPM_k(n)$.*

Lemma 9 also allows to deduce that for all unimodal sequence a, the set $S(a)$ of integers j such that (a, j) is in $\mathrm{SIPM}_k(n)$ is an integer interval.

Corollary 3. *Given $a \in US(n)$, let j_1, j_2 be two integers such that $(a, j_1), (a, j_2) \in SIPM_k(n)$. Then for all j with $j_1 \leq j \leq j_2$ one has $(a, j) \in SIPM_k(n)$.*

Our algorithm generates all possible unimodal sequences a that are form of some symmetric ice pile, and, for each form, all integers j such that $(a, j) \in \mathrm{SIPM}_k(n)$. We only need to determine the bounds of $S(a)$.

Corollary 4. *Let $a \in US(n)$ Then $(a, j) \in SIPM_k(n)$ if and only if $j \in [j_{\min}, j_{\max}]$ with $j_{\min} = \min\{j | (a, j) \in SIPM_k(n)\}$ and $j_{\max} = \max\{j | (a, j) \in SIPM_k(n)\}$.*

For the computation of the two integers $j_{\min}$, $j_{\max}$ we have:

Corollary 5. *Let $a \in US(n)$ Then $j_{\min} = \min\{j | (a, j) \in SIPM_k(n)\}$ and $j_{\max} = \max\{j | (a, j) \in SIPM_k(n)\}$ are computed in time $O(1)$.*

Example 2. Let $n = 21$, $k = 2$ and $a = (1, 2, 2, 3, 4, 4, 4, 1)$ with $D(a) = [i_1, i_2] = [4, 6]$. Clearly $[j_{\min}, j_{\max}] \supseteq [-i_2, -i_1] = [-6, -4]$ because $(a, -6)$ and $(a, -4)$ are decomposable at 0. However the largest $\delta > 0$ such that $(a, -4 + \delta) \in \mathrm{SIPM}_2(21)$ is 1. Indeed, $(a, -3)$ is decomposable at 1 and $w((a, -3), 1) = 18 < 21$, whereas $(a, -2)$ is decomposable only at j, $2 \leq j \leq 4$, but with weight $w((a, -2), 2) = 23 > 21$. On the other hand, $(a, -7) \notin \mathrm{SIPM}_2(21)$ since it is decomposable only at j, $-3 \leq j \leq -1$ with $w((a, -7), -1) = 25 > 21$. Therefore $(a, j) \in \mathrm{SIPM}_2(21)$ if and only if $j \in [-6, -3]$.

We conclude this section with a slightly different characterization of forms of symmetric ice piles which is more suitable for our generation algorithm.

Lemma 10. *A unimodal sequence a is the form of an element in $SIPM_k(n)$ if and only if $a = c \cdot x^{[p]} \cdot d$ with $\bar{c} \in IPM_k(t_1), d \in IPM_k(t_2)$, $h(\bar{c}), h(d) < x$, $t_1+t_2+xp = n$ and either $p < 2k+1$ (1) or $p = 2k+2$ and $\bar{c}, d$ are not x-critical (2) or $p = 2k+1$ and at least one of $\bar{c}, d$ is not x-critical (3).*

Given a unimodal sequence $a = c \cdot x^{[p]} \cdot d$ that is the form of an element in $\mathrm{SIPM}_k(n)$, the triple $(x, p, w(d))$ is said the *type* of a (by Lemma 10 $p \leq 2k+2$). In other words, a triple of integers (x, y, z) is a type for $\mathrm{SIPM}_k(n)$ if and only if there are a unimodal sequence $a = c \cdot x^{[y]} \cdot d$, with $\bar{c} \in \mathrm{IPM}_k(n - xy - z)$ and $d \in \mathrm{IPM}_k(z)$, and an integer j such that $(a, j) \in \mathrm{SIPM}_k(n)$. Obviously one has $\sqrt{n} - 1 \leq x \leq n$, $1 \leq p \leq \min(\lfloor n/x \rfloor, 2k+2)$ and $0 \leq r \leq kx(x-1)/2 + x - 1$.

4 The algorithm

The idea is that of generating in order (with respect to $<$) all unimodal sequences that are forms of elements in $\mathrm{SIPM}_k(n)$. By Lemma 10, such forms are the product of three sequences, $c \cdot x^{[p]} \cdot d$, which satisfy suitable constraints. Thus, we consider all the triples (x, p, r) corresponding to types, then we generate all the forms associated with each type, and lastly, for each form, we compute all the positions of the symmetric ice piles having that form. So, consider a type (x, p, r) and distinguish three cases depending on p.

When $p \leq 2k$ all the forms can be written as $a = c \cdot x^{[p]} \cdot d$ where $\bar{c}$ and d are arbitrary ice piles in $\mathrm{IPM}_{k,x-1}(n - px - r)$ and $\mathrm{IPM}_{k,x-1}(r)$, respectively. In fact, $x^{[j_1]} \cdot \bar{c}$ and $x^{[j_2]} \cdot d$ are always ice piles for all $j_1, j_2 \leq k$. By choosing two values such that $j_1 + j_2 = p$ we can guarantee the existence of a value i such that $\overline{a_{<i}}$ and $a_{\geq i}$ are ice piles. Then, Lemma 5 states that the unimodal sequence $c \cdot x^{[p]} \cdot d$ is the form of an element in $\mathrm{SIPM}_k(n)$.

Obviously, $c \cdot x^{[p]} d \neq c' \cdot x^{[p]} \cdot d'$ if $h(c), h(d), h(c'), h(d') < x$ and $c' \neq c$ or $d' \neq d$. Thus, we get all the forms of this type, if we generate all the elements of $\mathrm{IPM}_{k,x-1}(r)$ and, for each of these, all the ice piles in $\mathrm{IPM}_{k,x-1}(n - px - r)$.

Consider the case $p = 2k+2$. In order to get all the forms $c \cdot x^{[2k+2]} \cdot d$, by condition (2) in Lemma 10 we need to generate all ice piles $\bar{c}$ and d that are not x-critical. By Corollary 1 these are the ice piles in $G(s)$ and $G(t)$, where s and t are the highest staircases in $\mathrm{IPM}_{k,x-1}(n-px-r)$ and $\mathrm{IPM}_{k,x-1}(r)$, respectively.

Lastly, let $p = 2k+1$. If $r < (k+1)(x-1)$ it is sufficient to generate all d in $\mathrm{IPM}_{k,x-1}(r)$ and (for each of these) all $\bar{c} \in \mathrm{IPM}_{k,x-1}(n-(2k+1)x-r)$. Observe that d is not x-critical and thus condition (3) of Lemma 10 is satisfied. Otherwise ($r \geq (k+1)(x-1)$), let t be the staircase of height $x-1$ in $\mathrm{IPM}_{k,x-1}(r)$ and consider the partition $A \cup B = \mathrm{IPM}_{k,x-1}(r)$ where $A = \{a \in \mathrm{IPM}_{k,x-1}(r) | a <_{\mathrm{nlex}} t\}$ and $B = \{b \in \mathrm{IPM}_{k,x-1}(r) | t \leq_{\mathrm{nlex}} b\}$. By Lemma 2 each element of A is x-critical and then for each $d \in A$ we have to generate all ice piles $\bar{c}$ that are not x-critical (so that $x^{[k+1]} \cdot \bar{c}$ is an ice pile): these are exactly the ice piles $\bar{c} \in G(s)$ where s is the staircase of height $x-1$ in $\mathrm{IPM}_{k,x-1}(n-(2k+1)x-r)$. Then, for each $d \in B$ we generate all $\bar{c} \in \mathrm{IPM}_{k,x-1}(n-(2k+1)x-r)$ ($x^{[k+1]} \cdot d$ is an ice pile). Observe that by Lemma 1 and Corollary 1 one has $A = G(g) \setminus G(t)$, where $g = \min_{<_{\mathrm{nlex}}}(\mathrm{IPM}_{k,x-1}(r))$, and $B = G(t)$.

Once a form $c \cdot x^{[p]} d$ has been generated, all the positions j such that $(c \cdot x^{[p]} \cdot d, j) \in \mathrm{SIPM}_k(n)$ are computed using Corollary 5.

This idea leads immediately to Algorithm 1. We represent ice piles in $\mathrm{IPM}_k(r)$ as structures with six fields: the number of grains r, the current length, the linear partition (an array of integers), the set of moves (an ordered stack of integers), the integer k and a flag which is *on* if and only if the ice pile is x-critical. This last field is included only to simplify the computation of the positions.

The algorithm consists of three nested loops (associated with the three entries of a type (x, p, r)). The repeat-loop is used to set the height x of the symmetric ice pile, starting from the initial value n. The for-loop sets the number p of columns of height x, from the the largest allowed value $\min(\lfloor n/x \rfloor, 2k + 2)$ downto 1. Lastly, the while-loop sets the weight r of the ice pile d in $c \cdot x^{[p]} \cdot d$, from the largest admissible value (the smallest between the number of available grains $n - px$ and either $kx(x-1)/2$ or $kx(x-1)/2 + x - 1$, depending on p) downto the smallest one (i.e. a value r such that (x, p, r) is a type for $\mathrm{SIPM}_k(n)$ while $(x, p, r - 1)$ is not).

Algorithm 1 Exhaustive Generation of $\mathrm{SIPM}_k(n)$.

```
1: PROCEDURE SIPGENERATION($n$,$k$)
2: $x := n$;
3: repeat
4:    for $p := \min(\lfloor n/x \rfloor, 2k+2)$ downto 1 do
5:       $m := n - p \cdot x$;
6:       if $p = 2k+2$ then $max := kx(x-1)/2$; else $max := kx(x-1)/2 + x - 1$;
7:       $r := \min(m, max)$;
8:       while ISTYPE($n$,$k$,$x$,$p$,$r$) do
9:          if $p < 2k+1$ then GEN1($m-r$,$r$,$x-1$,$p$,$k$);
10:                          else if $p = 2k+2$ then GEN2($m-r$,$r$,$x-1$,$k$);
11:                          else if $p = 2k+1$ then GEN3($m-r$,$r$,$x-1$,$k$);
12:         $r := r - 1$;
13:      end while
14:   end for
15:   $x := x - 1$;
16: until TOOLOW($n$,$k$,$x$);
```

At each iteration, the value p determines which case of Lemma 10 occurs.

When $p < 2k+1$ the call GEN1($m-r$,r,$x-1$,p,k) generates all symmetric ice piles having the form $c \cdot x^{[p]} \cdot d$ with $\bar{c} \in \mathrm{IPM}_{k,x-1}(m-r)$ and $d \in \mathrm{IPM}_{k,x-1}(r)$. Similarly, if $p = 2k+2$ GEN2($m-r$,r,$x-1$,k) generates all symmetric ice piles having the form $c \cdot x^{[2k+2]} \cdot d$ with $\bar{c} \in G(s)$, and $d \in G(t)$, where s and t are the highest staircases in $\mathrm{IPM}_{k,x-1}(m-r)$ and $\mathrm{IPM}_{k,x-1}(r)$, respectively. Lastly, when $p = 2k+1$, the call GEN3($m-r$,r,$x-1$,k) generates all symmetric ice piles having the form $c \cdot x^{[2k+1]} \cdot d$ and such that $\bar{c} \in \mathrm{IPM}_{k,x-1}(m-r)$ or $d \in \mathrm{IPM}_{k,x-1}(r)$ is not x-critical.

The algorithm halts as soon as a value x is reached such that no symmetric ice pile of height x exists (TooLow(n,k,x) at line 24 returns true iff $\mathrm{IPM}_{k,x}(n) = \emptyset$).

Procedures Gen [1-3] are easily developed by means of the following functions and procedures. MinIcePile(x,r,k) costructs the smallest ice pile in $\mathrm{IPM}_{k,x}(r)$ while Staircase(x,r,k) constructs the highest staircase in $\mathrm{IPM}_{k,x}(r)$. Both functions return a reference to an ice pile (the former possibly sets the flag indicating x-criticality). IsStaircase(a) (IsBottom(a)) returns the boolean value *true* iff a is a staircase (fixed point). The generation of the ice piles $\bar{c}$ and d which appear in $c \cdot x^{[p]} \cdot d$ is done by means of Next. This function sets its argument to the next ice pile in the neglex order, and returns a reference to it (see Section 2.1). Once the components c, x, p and d of a form are known, the Procedure Positions computes the range of positions for that form.

Example 3. SIPGeneration$(7, 2)$ produces the following sequence of symmetric ice piles (see Fig. 4): $(7), (6)1, 1(6), (5)2, (5)11, 1(5)1, \ldots, 111(2)11, 11(1)211$.

5 Complexity

With respect to the space complexity, we point out that the algorithm uses $O(\sqrt{kn})$ space, since to represent a form $c \cdot x^{[p]} \cdot d$ we need only two ice piles $\bar{c}$, d (represented by two structures of size $O(\sqrt{kn})$) and two integers x, p. Moreover, recall that for any $a \in \mathrm{IPM}_k(n)$ the generation of $G(a)$ requires $O(|G(a)|)$ time and $O(\sqrt{kn})$ space (see section 2.1).

Procedures MinIcePile and Staircase have a cost which grows as the length $l = O(\sqrt{kn})$ of the returned ice pile, whereas IsBottom and IsStaircase run in time $O(1)$. Indeed, these two functions simply check the stack ST representing the moves of the ice pile ($\mathrm{ST} = \emptyset, \mathrm{ST} \subseteq \{l-2, l-1\}$).
Functions IsType and TooLow run in time $O(1)$ too. In fact, IsType(n, k, x, p, r) when $p \leq 2k$ simply checks whether the two values r and $n - px - r$ are both smaller than $kx(x-1)/2 + x$. Similarly, if $p = 2k+2$ it verifies that $r, n - (2k+2)x - r \leq kx(x-1)/2$. Lastly, if $p = 2k+1$ it checks whether $r \leq kx(x-1)/2 + x - 1$ and $n - (2k+1)x - r \leq kx(x-1)/2$ or $r \leq kx(x-1)/2$ and $n - (2k+1)x - r \leq kx(x-1)/2 + x - 1$. TooLow$(n, k, x)$ returns *false* if $2kx(x-1)/2 \leq n - (2k+2)x$, *true* otherwise. At last, Corollary 5 lets us develop a procedure Positions$(\bar{c}, x, p, d)$ which runs in time $O(1)$.

In Procedure SIPGeneration, the repeat-loop iterates $O(n)$ times, the for-loop iterates $O(k)$ times and the while-loop iterates $O(kn^2)$ times. Then the overall cost is $O(k^2n^3)$ plus the cost of $O(k^2n^3)$ calls to Gen[1-3]. Thus, we consider:

Lemma 11. *Let $C(l, r, x, p, k)$ be the number of symmetric ice piles generated by either* Gen1*(l,r,x,p,k) (if $p \leq 2k$) or* Gen3*(l,r,x,k) (if $p = 2k+1$) or* Gen2*(l,r,x,k) (if $p = 2k+2$). Then, the running time of* Gen1*(l,r,x,p,k),* Gen2*(l,r,x,k) and* Gen3*(l,r,x,k) is $O(C(l, r, x, p, k)) + O(\sqrt{kn})$.*

We can now prove the main result.

Theorem 1. SIPGENERATION *is a CAT algorithm and uses* $O(\sqrt{kn})$ *space.*

Proof. Note that all the instructions of SIPGENERATION have cost $O(1)$ except the calls to GEN[1-3]. Thus, the cost $T(n)$ of SIPGENERATION(n,k) is $O(k^2n^3)$ (due to the three nested loops) plus the cost of $O(k^2n^3)$ calls to GEN[1-3]. Therefore, by Lemma 11 one has

$$T(n) = O(k^2n^3) + \sum_{x,p,l,r} O(C(l,r,x,p,k)) = O(k^2n^3) + O(|\text{SIPM}_k(n)|).$$

Since $k^2n^3 = O(|\text{SPM}(n)|)$ and $|\text{SPM}(n)| \leq |\text{IPM}_k(n)| \leq |\text{SIPM}_k(n)|$ (see [3] for bounds for $|\text{SPM}(n)|$), SIPGENERATION turns out to be a CAT algorithm. With respect to the space complexity, note that the necessary space for storing two ice piles at a time is $(O(\sqrt{kn}))$. □

As an extension of this work, it is quite natural to ask whether a similar approach can be applied to deal with the exhaustive generating problem for other (similar) discrete models. In particular, the discrete models BSPM and BIPM (Bidimensional Sand and Ice Pile Model) have been introduced in [4] by adding a further dimension to SPM(n) and IPM$_k$(n), respectively. Thus, the elements of BSPM and BIPM are plane partitions, that is, matrices of nonnegative integers that are nonincreasing from top to bottom and from left to right. These models are not lattices and admit several fixed points but, unlike SIPM$_k$(n), no characterization is known for reachable states and fixed points.

References

1. P. Bak, C. Tang and K. Wiesenfeld, Self-organized criticality, *Phys. Rev. A* 38 (1988), 364–374.
2. T. Brylawski, The lattice of integer partitions, *Discrete Math.* 6 (1973), 201–219.
3. S. Corteel, D. Gouyou-Beauchamps, Enumeration of sand piles, *Discrete Math.* 256 n.3 (2002), 625–643.
4. E. Duchi, R. Mantaci, H. D. Phan and D. Rossin, Bidimensional sand pile and ice pile models, *PU.M.A.* vol. 17 (2007) n.1-2, 71–96.
5. E. Formenti, B. Masson and T. Pisokas, Advances in symmetric sandpiles, *Fundamenta Informaticae* 20 (2006), 1–22.
6. E. Goles and M. A. Kiwi, Games on line graphs and sand piles, *Theoret. Comput. Sci.* 115 (1993), 321–349.
7. E. Goles, M. Morvan and H. D. Phan, Sandpiles and order structure of integer partitions, *Discrete Appl. Math.* 117 (2002), 51–64.
8. M. Latapy, R. Mantaci, M. Morvan and H. D. Phan, Structure of same sand piles model, *Theoret. Comput. Sci.* 262 (2001), 525–556.
9. P. Massazza, A CAT algorithm for sand piles, *PU.M.A.* vol. 19 (2008) n.2-3, 147–158.
10. P. Massazza and R. Radicioni A CAT algorithm for the exhautive generation of ice piles, *RAIRO* Informatique théorique 44 (2010), 525–543.
11. P. Massazza, On the Exhaustive Generation of Symmetric Sand Piles, *Proc. of GASCom 2010*, Montréal, 2-4 September 2010.
12. H. D. Phan, Two sided sand piles model and unimodal sequences, *RAIRO* Informatique théorique 42 (2008), 631–646.

Adding two equivalence relations to the interval temporal logic AB

Angelo Montanari[1], Marco Pazzaglia[1], and Pietro Sala[2]

[1] Department of Mathematics and Computer Science,
University of Udine, Italy
angelo.montanari@uniud.it, marco@pazzaglia.me
[2] Department of Computer Science
University of Verona, Italy
pietro.sala@univr.it

Abstract. The interval temporal logic AB features two modalities that make it possible to access intervals which are adjacent to the right of the current interval (modality $\langle A \rangle$) and proper subintervals that have the same left endpoint of it (modality $\langle B \rangle$). AB is one of the most significant interval logics, as it allows one to express meaningful (metric) properties, while maintaining decidability (undecidability rules over interval logics, AB is EXPSPACE-complete [14]). In an attempt to capture ωS-regular languages with interval logics [15], it was proved that AB extended with an equivalence relation, denoted AB$\sim$, is decidable (non-primitive recursive) on the class of finite linear orders and undecidable on $\mathbb{N}$. The question whether the addition of two or more equivalence relations makes finite satisfiability for AB undecidable was left open. In this paper, we answer this question proving that AB$\sim_1\sim_2$ is undecidable.

1 Introduction

Interval temporal logics (ITL) are temporal logics where time intervals/periods, instead of time points/instants as in the standard framework, are used as basic building blocks. ITL are characterized by high expressiveness and high computational complexity. The main formalization of these logics is known as HS which features one modality for each interval order relation [7] (the so-called Allen's relations [1]). In this paper, we analyze the complexity of the finite satisfiability problem for the interval temporal logic AB of Allen's relations *meets* and *begun by* extended with two equivalence relations. AB is one of the most significant fragments of HS since it is decidable [14] (undecidability rules over interval logics [5]) and it can express various important (metric) temporal properties. As an example, it allows one to encode the standard until operator of point-based linear temporal logic as well as to constrain the length of an interval to be less than/equal to/greater than a given value [14].

The trade-off between the increase in expressiveness and the complexity blow-up induced by the addition of one or more equivalence relations to a logic has been already highlighted in the literature (see, for instance, the logics for semi-structured data [3], temporal logics [6], and timed automata [16]). Finite satisfiability of the two-variable fragment of first-order logic FO^2 extended with one, two, or more equivalence relations has been systematically explored in [8–10], while the extension of FO^2, interpreted over finite or infinite data words, with an equivalence relation has been investigated by Bojańczyk et al. in [3]. Similar results have been obtained by Demri and Lazic [6], that studied the extension of linear temporal logic over data words with freeze quantifiers, which allow one to store elements at the current word position into a register and then to use them in equality comparisons deeper in the formula, and by Ouaknine and Worrell [16], who showed that both satisfiability and model checking for metric temporal logic over finite timed words are decidable with a non-primitive recursive complexity.

The addition of an equivalence relation to an interval temporal logic was first investigated by Montanari and Sala in [15]. They focused their attention on the interval logic AB of Allen's relations *meets* and *begins* extended with an equivalence relation, denoted $\mathsf{AB}{\sim}$, interpreted over finite linear orders and $\mathbb{N}$, and they showed that the resulting increase in expressive power makes it possible to establish an original connection between interval temporal logics and extended regular languages of finite and infinite words [2]. As for the computational complexity, they proved that $\mathsf{AB}{\sim}$ is decidable (non-primitive recursive) on the class of finite linear orders and undecidable on $\mathbb{N}$. Recently, the interval logic of temporal neighborhood PNL, which features two modalities for Allen's relations *meets* and *met by*, and its metric variant MPNL, both extended with one equivalence relation, have been proved to be decidable over finite linear orders [13] (NEXPTIME-complete the former, EXPSPACE-hard the latter). In this paper, we answer a question left open in [15], showing that the addition of two (or more) equivalence relations makes AB undecidable also over finite linear orders.

The paper is organized as follows. Section 2 illustrates in some detail previous work on the interval temporal logic AB extended with one equivalence relation and some related work. Section 3 introduces syntax and semantics of the logic $\mathsf{AB}{\sim_1}{\sim_2}$ and gives some background knowledge about counter machines. The next two sections provide a reduction from the undecidable 0-0 reachability problem for Minsky counter machines to the finite satisfiability problem for $\mathsf{AB}{\sim_1}{\sim_2}$. More precisely, Section 4 outlines the general structure forced on each model (if any) by the formulas given in Section 5. Finally, in Section 6 we prove that the proposed encoding is correct. Conclusions provide an assessment of the work and outline future research directions.

2 Related work

The present paper can be viewed as the natural completion of the work reported in [15], where Montanari and Sala proved that the satisfiability problem for $\mathsf{AB}{\sim}$ is decidable over finite linear orders, with non-primitive recursive complexity, and undecidable over $\mathbb{N}$. Undecidability has been proved by a reduction from the (undecidable) 0-n reachability problem for lossy counter machines [11]. As for finite satisfiability, they initially reduced the problem of finding a model for a given $\mathsf{AB}\sim$ formula φ to the existence of a particular *compass structure* exploiting the correspondence that can be established between intervals and points in the positive octant of the Cartesian plane, that is, the map that links any interval $[x, y]$ to the corresponding point (x, y). Then, by exploiting a suitable model contraction technique, they showed that if φ is finitely satisfiable, then a structure satisfying it can be obtained via a bottom-up generation of candidate (finite) compass structures. Since the number of pairwise incomparable rows of any candidate structure can be proved to be finite, termination easily follows. The complexity bound has been obtained by encoding in $\mathsf{AB}\sim$ the 0-0 reachability for *lossy* counter machines (LCM) which is known to be a non-primitive recursive (decidable) problem [17].

The general structure of the $\mathsf{AB}\sim$ encoding of the 0-0 reachability problem for LCM is similar to the one provided in this paper for (non-lossy) counter machines (CM). However, when only one equivalence relation is available, it is possible to enforce the presence of certain points in a configuration (depending on the points of the previous configuration), but not to restrict the number of points in it. This last constraint is necessary for the encoding of CM and it can be enforced only by making use of two equivalence relations.

From a technical point of view, our work presents some similarities to the one by Bojańczyk et al. in [4], where it is shown, among other results, that the logic $\mathsf{FO}^2(+1, \sim_1, \sim_2)$ over finite data words is undecidable. To prove it, they provide a reduction from the Post correspondence problem to finite satisfiability for $\mathsf{FO}^2(+1, \sim_1, \sim_2)$, that exploits the interconnections between equivalence relations in a way that is similar to what we do here. However, their encoding strongly depends on constraints of the form $\forall\exists$ which are not expressible in AB. To overcome these limitations, we will exploit some metric properties definable in AB.

3 Preliminaries

In this section, we first introduce syntax and semantics of $\mathsf{AB}\sim_1\sim_2$ and then we provide background knowledge about Minsky counter machines.

3.1 The interval temporal logic $\mathsf{AB}\sim_1\sim_2$

The interval temporal logic $\mathsf{AB}\sim_1\sim_2$ features two modalities $\langle \mathsf{A} \rangle$ and $\langle \mathsf{B} \rangle$ corresponding to Allen's relations *meets* and *begun by*, respectively, and two special binary relation symbols $\sim_1$ and $\sim_2$. Formally, given a set $\mathcal{P}rop$ of propositional variables, formulas of $\mathsf{AB}\sim_1\sim_2$ are built up from $\mathcal{P}rop$ and $\sim_1, \sim_2$ using the Boolean connectives $\neg$, $\vee$ and the modalities $\langle \mathsf{A} \rangle$ and $\langle \mathsf{B} \rangle$. Moreover, we make use of shorthands $\varphi_1 \wedge \varphi_2$ for $\neg(\neg\varphi_1 \vee \neg\varphi_2)$, $[\mathsf{A}]\varphi$ for $\neg\langle \mathsf{A} \rangle\neg\varphi$, $[\mathsf{B}]\varphi$ for $\neg\langle \mathsf{B} \rangle\neg\varphi$, $\top$ for $p \vee \neg p$, and $\bot$ for $p \wedge \neg p$, with $p \in \mathcal{P}rop$.

We interpret formulas of $\mathsf{AB}{\sim_1}{\sim_2}$ in interval temporal structures over (prefixes of) $\mathbb{N}$ endowed with the ordering relations *meets* (denoted by A) and *begun by* (denoted by B), and two equivalence relations $\sim_1$ and $\sim_2$. More precisely, we identify any given ordinal $N < \omega$ with the prefix of length N of $\mathbb{N}$ and we accordingly define $\mathbb{I}(N)$ as the set of all closed intervals $[i, j]$, with $i, j \in N$ and $i \leq j$. A special role will be played by point-intervals (intervals of the form $[i, i]$, with $i \in N$) and unit-intervals (intervals of the form $[i, i + 1]$), which can be respectively defined as $[\mathsf{B}] \perp$ (abbreviated π) and $[\mathsf{B}][\mathsf{B}] \perp$ (abbreviated *unit*). For any pair of intervals $[i, j], [i', j'] \in \mathbb{I}(N)$, relations A and B are defined as follows:

– *meets* relation: $[i, j]\ A\ [i', j']$ iff $j = i'$;
– *begun by* relation: $[i, j]\ B\ [i', j']$ iff $i = i'$ and $j' < j$.

The (non-strict) semantics of $\mathsf{AB}{\sim_1}{\sim_2}$ is given in terms of interval models $\mathcal{S} = \langle \mathbb{I}(N), A, B, \sim_1, \sim_2, V\rangle$, where $\sim_1$ and $\sim_2$ are two equivalence relations over N and $V\colon \mathbb{I}(N) \to \wp(\mathcal{P}rop)$ is a valuation function that assigns to every interval $[i, j] \in \mathbb{I}(N)$ the set of propositional variables $V([i, j])$ that are true on it. The truth of an $\mathsf{AB}{\sim_1}{\sim_2}$ formula over a given interval $[i, j]$ in a model $\mathcal{S}$ is defined by structural induction as follows:

– $\mathcal{S}, [i, j] \models p$ iff $p \in V([i, j])$, for all $p \in \mathcal{P}rop$;
– $\mathcal{S}, [i, j] \models \neg\psi$ iff $\mathcal{S}, [i, j] \not\models \psi$;
– $\mathcal{S}, [i, j] \models \varphi \vee \psi$ iff $\mathcal{S}, [i, j] \models \varphi$ or $\mathcal{S}, [i, j] \models \psi$;
– $\mathcal{S}, [i, j] \models \langle \mathsf{X}\rangle\psi$ iff there exists an interval $[i', j']$ such that $[i, j]\ X\ [i', j']$, and $\mathcal{S}, [i', j'] \models \psi$, for $X \in \{A, B\}$;
– $\mathcal{S}, [i, j] \models \sim_k$ iff $i \sim_k j$, for $k \in \{1, 2\}$.

Given an interval structure $\mathcal{S}$ and a formula φ, we say that $\mathcal{S}$ *satisfies* φ if there is an interval I in $\mathcal{S}$ such that $\mathcal{S}, I \models \varphi$. We say that φ is *(finitely) satisfiable* if there exists a (finite) interval structure that satisfies it. We define the *(finite) satisfiability problem* for $\mathsf{AB}{\sim_1}{\sim_2}$ as the problem of establishing whether a given $\mathsf{AB}{\sim_1}{\sim_2}$ formula φ is (finitely) satisfiable.

3.2 Counter machines

A k *counter machine* (kCM) is a triple of the form $M = (Q, k, \delta)$, where Q is a finite set of control states, k is the number of counters, whose values range over $\mathbb{N}$, and δ is a function that maps each state $q \in Q$ to a transition rule having one of the following forms:

1. $\mathit{value}(h) \leftarrow \mathit{value}(h) + 1;\ \mathit{goto}\ q'$, for some $1 \leq h \leq k$ and $q' \in Q$ (abbreviated $(q, h{+}{+}, q')$), meaning that, whenever M is at state q, it increases the counter h and it moves to state q';
2. if $\mathit{value}(h) = 0$ then goto q' else $\mathit{value}(h) \leftarrow \mathit{value}(h) - 1$; goto q'', for some $1 \leq h \leq k$ and $q', q'' \in Q$ (abbreviated $(q, h?0, q', q'')$), meaning that, whenever M is at state q and the value of the counter h is 0 (resp., greater than 0), it moves to state q' (resp., it decrements the counter h and it moves to state q'').

A computation of M is any sequence of configurations that conforms to the transition relation. The reachability problem for a counter machine $M = (Q, k, \delta)$ is the problem of deciding, given two configurations $(q_0, \overline{z}_0)$ and $(q_f, \overline{z}_f)$, whether there is a computation that takes M from $(q_0, \overline{z}_0)$ to $(q_f, \overline{z}_f)$. The reachability problem for counter machines is undecidable even for machines with only two counters [12]. For convenience, we will use a restricted, but equally undecidable, form of this problem, called 0-0 *reachability* problem, where z_0 and z_f are both $\overline{0}$. Moreover, without loss of generality, we restrict our attention to computations where q_0 and q_f occur only at the beginning and at the end, respectively.

4 The structure of intended models

The main contribution of the paper is the proof of the following theorem.

Theorem 1. *The satisfiability problem for* $\mathsf{AB}{\sim_1}{\sim_2}$ *over the class of finite linear orders is undecidable.*

To prove it, we provide a reduction from the 0-0 reachability problem for Minsky two-counter machines to the satisfiability problem for $\mathsf{AB}{\sim_1}{\sim_2}$. More precisely, given a two-counter machine $M = (Q, 2, \delta)$ and two states $q_0, q_f \in Q$, we build a formula ψ_{M,q_0,q_f} such that there exists a computation in M from the configuration $(q_0, 0, 0)$ to the configuration $(q_f, 0, 0)$ if and only if ψ_{M,q_0,q_f} is satisfiable. First, in the present section, we delineate the structure that we want to give to each model (if any) of ψ_{M,q_0,q_f} (the intended model). Then, in Section 5, we show how to encode in $\mathsf{AB}{\sim_1}{\sim_2}$ such a structure. We conclude the paper with the proof of the correctness of the encoding.

The structure of the models of the encoding formula can be described as follows. To start with, we partition the set of point-intervals (points for short) in two subsets: points labeled by a state in Q (*state-points*) and points labeled by c_1 or c_2 (*counter-points*). A configuration (q, v_1, v_2) is represented as a set of contiguous points, where the first point is a state-point with label q and the remaining ones are counter-points such that exactly v_1 points have label c_1 and v_2 points have label c_2, in any order. Counter-points with label *del* are points which have been "deleted" and thus do not count for the value of a counter in a configuration.

The computation of the two-counter machine M (from q_0 to q_f) consists of a sequence of contiguous configurations. Counter-points with label *plus* and *minus* indicate, respectively, points added in a configuration as a result of a counter increment or eliminated in the next configuration as a result of a counter decrement. Each configuration, but the first one, is obtained from the previous one by applying a transition of M, which amounts to say that the sequence of configurations is a valid computation of M.

In Figure 1, we give an example of the proposed model representation for the computation $(q, 1, 1) \to (q', 1, 0) \to (q'', 2, 0)$. The second configuration $(q', 1, 0)$ is obtained from the initial configuration $(q, 1, 1)$ by decrementing the second counter; the third configuration is obtained from the second one by incrementing the first counter. Notice that points with label *minus* are deleted, that is, labeled by *del*, in the configuration that immediately follows the one in which the decrement of the counter takes place.

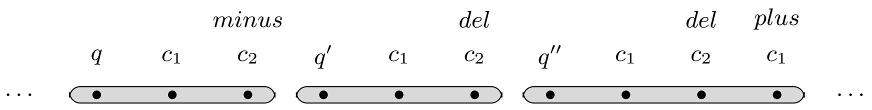


Fig. 1. Labels of the points in a model representing the partial computation $(q, 1, 1) \to (q', 1, 0) \to (q'', 2, 0)$.

5 Encoding of 0-0 reachability in AB$\sim_1\sim_2$

In this section, we show how to encode intended models, representing valid computations of the two-counter machine, by an AB$\sim_1\sim_2$ formula.

Let us consider the AB$\sim_1\sim_2$ formula ψ_{M,q_0,q_f}, which is defined as follows:

$$\psi_{M,q_0,q_f} \equiv \psi_{0\to 0} \wedge [\mathsf{G}](\psi_{points} \wedge \psi_\delta \wedge \psi_\sim),$$

where $[\mathsf{G}]\varphi$ is an shorthand for $[\mathsf{B}]\varphi \wedge \varphi \wedge [\mathsf{B}][\mathsf{A}]\varphi \wedge [\mathsf{A}][\mathsf{A}]\varphi$ (universal modality) and

- $\psi_{0\to 0}$ forces the initial and final configurations of the computation to be respectively $(q_0, 0, 0)$ and $(q_f, 0, 0)$;
- ψ_{points} specifies conditions on points: (i) points are partitioned in state-points and counter-points, (ii) labels *plus* and *minus* can label counter-points only, and (iii) in a configuration, there is at most one point with label *minus* and at most one point with label *plus*;
- ψ_δ ensures the consistency of state-points and *plus*/*minus* points with the transitions in M, that is, if a configuration τ', with state-point q', immediately follows a configuration τ, with state-point q, then one of the following three cases must hold: (i) there is a transition $\delta_i = (q, h++, q') \in \delta$, there is no point with label *minus* in τ, and there is one point with labels c_h and *plus* in τ'; (ii) there is a transition $\delta_i = (q, h?0, q', q'')$, there is no point with label c_h and without label *del* in τ, and there is no point with label *plus* in τ'; (iii) there is a transition $\delta_i = (q, h?0, q'', q')$, there is one point with label c_h and without label *del* in τ, and there is no point with label *plus* in τ';
- $\psi_\sim$ guarantees that each configuration is obtained from the previous one by an increment/decrement/no-action transition (notice that the fact that the transition actually belongs to M is checked by ψ_δ and not by $\psi_\sim$).

Formulas $\psi_{0\to 0}$, ψ_{points}, and ψ_δ can be easily expressed in AB, that is, equivalence relations play no role in the encoding of the corresponding conditions.

The main component of the encoding is the formula $\psi_{\sim}$, which acts like a controller of the form of configurations by preventing the addition of any unwanted counter-point. More precisely, it forces each configuration to be an isomorphic copy of the previous configuration, that is, to feature the same counter-points in the same order, plus at most one extra point with label *plus*.

We now show how to express each of above conditions in $\mathsf{AB}{\sim_1}{\sim_2}$. To facilitate the reading of the formulas, we make use of the following abbreviations: we denote the formula $\bigvee_{i=1}^{n} q_i$ by the symbol q and the formula $\bigvee_{i=1}^{2} c_i$ by the symbol c. Moreover, we define the following formula, parametric in φ:

$$succ(\varphi) \equiv \langle \mathsf{A} \rangle (unit \wedge \langle \mathsf{A} \rangle (\pi \wedge \varphi))$$

which states the truth of φ at the point immediately after the current interval, that is, at the successor of the right endpoint of the current interval. Finally, we introduce a derived modality $\langle \mathsf{P} \rangle$, which is defined in terms of modalities $\langle \mathsf{A} \rangle$ and $\langle \mathsf{B} \rangle$, that allows one to force a given formula φ to be true at some point of the current interval, endpoints included. The modality $\langle \mathsf{P} \rangle$ is formally defined as follows:

$$\langle \mathsf{P} \rangle \varphi \equiv \langle \mathsf{B} \rangle \langle \mathsf{A} \rangle (\pi \wedge \varphi) \vee \langle \mathsf{A} \rangle (\pi \wedge \varphi).$$

The dual of the above modality is defined as usual, that is, $[\mathsf{P}]\varphi \equiv \neg \langle \mathsf{P} \rangle \neg \varphi$, and it states that φ holds at each point of the current interval (endpoints included).

The initial and final configurations are encoded by the formula:

$$\psi_{0 \to 0} \equiv q_0 \wedge succ(q) \wedge \langle \mathsf{A} \rangle \langle \mathsf{A} \rangle q_f \wedge [\mathsf{A}][\mathsf{A}](q_f \to succ([\mathsf{A}][\mathsf{A}](\neg q \wedge (c \to del)))).$$

It states that the initial configuration is $(q_0, 0, 0)$ (the first state-point q_0 is followed by a state-point) and that the final configuration is $(q_f, 0, 0)$ (the last state-point is q_f and every counter-point after it, if any, is deleted).

The formula ψ_{points} is defined as the conjunction of the following conditions (hereafter, we explicitly provide the encoding of the most complex conditions only):

(A1) every point of the domain has one and only one label from the set $Q \cup \{c_1, c_2\}$;

(A2) labels in $Q \cup \{c_1, c_2\} \cup \{plus, minus, last, del\}$ are given to points only;

(A3) at most one counter-point per configuration can be labeled with *plus* and at most one with *minus*;

(A4) the label *last* is associated with the points of the final configuration only:

$$(last \to \pi) \wedge (last \leftrightarrow [\mathsf{A}](\neg \pi \to [\mathsf{A}] \neg q));$$

(A5) only counter-points can be labeled with *del* and a point can not have both label *del* and label *plus* (or *minus*).

The formula ψ_δ is defined as the conjunction of the following conditions:

(B1) all configurations but the initial one (and only them) have one (and only one) label in $\delta = \{\delta_1, ..., \delta_m\}$. We label the first configuration with a dummy transition δ_0. If a configuration is labeled with δ_i, for some $i > 0$, it means that it is obtained from the previous configuration by the application of the transition δ_i;

(B2) transition labels are consistent with state-points of each configuration:

$$\begin{aligned}&\bigwedge_{\delta_i=(q,h{+}{+},q')\in\delta} (\delta \wedge succ(\langle\mathsf{A}\rangle\delta_i)) \rightarrow \\ &\qquad\qquad (\langle\mathsf{B}\rangle q \wedge succ(q' \wedge \langle\mathsf{A}\rangle(\delta_i \wedge \langle\mathsf{P}\rangle(c_h \wedge plus)))) \\ \wedge &\bigwedge_{\delta_i=(q,h?0,q',q'')\in\delta} (\delta \wedge [\mathsf{P}](c_h \rightarrow del) \wedge succ(\langle\mathsf{A}\rangle\delta_i)) \rightarrow (\langle\mathsf{B}\rangle q \wedge succ(q')) \\ \wedge &\bigwedge_{\delta_i=(q,h?0,q',q'')\in\delta} (\delta \wedge \langle\mathsf{P}\rangle(c_h \wedge \neg del) \wedge succ(\langle\mathsf{A}\rangle\delta_i)) \rightarrow \\ &\qquad\qquad (\langle\mathsf{B}\rangle q \wedge succ(q'') \wedge \langle\mathsf{P}\rangle(c_h \wedge minus)),\end{aligned}$$

where δ is a shorthand for the formula $\bigvee_{i=0}^{m} \delta_i$;

(B3) there are no points labeled with *plus* in configurations labeled with non-increment transitions or point labeled with *minus* in configurations that precede configurations labeled with non-decrement transitions;

(B4) every (non-final) configuration devoid of counter-points is followed by a configuration with at most one counter-point (labeled with *plus*).

In order to define the formula $\psi_\sim$, it turns out to be useful to introduce the following formula:

$$\psi_{\exists! q} \equiv ([\mathsf{B}](\langle\mathsf{A}\rangle q \rightarrow [\mathsf{B}]\langle\mathsf{A}\rangle\neg q) \wedge \langle\mathsf{B}\rangle\langle\mathsf{A}\rangle q \wedge \langle\mathsf{A}\rangle\neg q) \vee (\langle\mathsf{A}\rangle q \wedge [\mathsf{B}][\mathsf{A}]\neg q),$$

which guarantees the existence of a unique state-point inside the current interval (endpoints included).

We call an interval labeled with both $\sim_1$ and $\sim_2$ and containing exactly one state-point a *linking* interval, that is, a linking interval is an interval that satisfies the formula $\sim_1 \wedge \sim_2 \wedge \psi_{\exists! q}$, and we say that its endpoints are *linked* together (the use of linking intervals in the encoding will be explained in Section 6).

The formula $\psi_\sim$ is defined as the conjunction of the following conditions:

(C1) a unit interval can not be labeled with both $\sim_1$ and $\sim_2$:

$$unit \rightarrow \neg(\sim_1 \wedge \sim_2);$$

(C2) there is no interval, whose endpoints belong to the same configuration, which is neither a point interval nor a unit interval and is labeled with $\sim_1$ or $\sim_2$:

$$(\neg\pi \wedge \neg unit \wedge [\mathsf{P}]\neg q) \rightarrow \neg(\sim_1 \vee \sim_2);$$

(C3) two consecutive counter-points are in relation $\sim_1$ or $\sim_2$:

$$(unit \wedge \langle \mathsf{B} \rangle c \wedge \langle \mathsf{A} \rangle c) \rightarrow (\sim_1 \vee \sim_2)$$

(C4) each counter-point, which does not belong to the last configuration, starts a linking interval:

$$(c \wedge \neg last) \rightarrow \langle \mathsf{A} \rangle (\sim_1 \wedge \sim_2 \wedge \psi_{\exists ! q});$$

(C5) the labels of the endpoints of a linking interval must satisfy the following constraints:

$$\begin{aligned}(\sim_1 \wedge \sim_2 \wedge \psi_{\exists ! q}) \rightarrow \big(& ((\langle \mathsf{B} \rangle c_1 \wedge \langle \mathsf{A} \rangle c_1) \vee (\langle \mathsf{B} \rangle c_2 \wedge \langle \mathsf{A} \rangle c_2)) \\ & \wedge (\langle \mathsf{B} \rangle del \rightarrow \langle \mathsf{A} \rangle del) \\ & \wedge (\langle \mathsf{B} \rangle minus \rightarrow \langle \mathsf{A} \rangle del) \\ & \wedge (\langle \mathsf{B} \rangle (\neg minus \wedge \neg del) \rightarrow \langle \mathsf{A} \rangle \neg del) \\ & \wedge (\langle \mathsf{A} \rangle \neg plus)\big);\end{aligned}$$

(C6) the first point of all configurations, but the final one, is linked to the first point of the next configuration:

$$\begin{aligned}&(unit \wedge \langle \mathsf{B} \rangle (q \wedge \neg last) \wedge \langle \mathsf{A} \rangle c) \\ &\qquad \rightarrow \langle \mathsf{A} \rangle (\sim_1 \wedge \sim_2 \wedge \psi_{\exists ! q} \wedge [\mathsf{B}](\langle \mathsf{A} \rangle q \vee [\mathsf{P}] \neg q)).\end{aligned}$$

We would like to observe that the formula $[\mathsf{B}](\langle \mathsf{A} \rangle q \vee [\mathsf{P}] \neg q))$ enforces the second to last point of the linking interval to be the only state-point. Let $[x, y]$ be this interval. If y', with $x < y' < y - 1$, were a state-point (it can not be the final point y since $[x, y]$ is a linking-interval, and, by C5, linking intervals have counter-points as their endpoints), then $[x, y - 1]$ would satisfy neither $\langle \mathsf{A} \rangle q$ (there can only be one state-point between x and y) nor $[\mathsf{P}] \neg q$ (since y' is between x and $y - 1$ and it is a state-point);

(C7) the last point of every configuration, but the final one, is linked to a point that is followed by a state-point or by a counter-point with label *plus* followed by a state-point:

$$\begin{aligned}(\delta_i \wedge \langle \mathsf{B} \rangle \neg last) \rightarrow \langle \mathsf{A} \rangle \big(& \sim_1 \wedge \sim_2 \wedge \psi_{\exists ! q} \wedge \\ & (succ(q) \vee succ(plus \wedge succ(q)))\big).\end{aligned}$$

We would like to emphasize the crucial role of condition (C5). First of all, it constrains linking intervals to connect counter-points with the same label (either c_1 or c_2). Moreover, it transfers logical deletion (counter-points labeled by *del*) from one configuration to the next one, it forces the actual execution of a new deletion by connecting a counter-point labeled by *minus* to a counter-point labeled by *del*, and it prevents unwanted deletions or insertions to take place.

6 Proof of correctness of the encoding

The proof of correctness for the formulas $\psi_{points}, \psi_\delta$, and $\psi_{0\to 0}$ is straightforward. The only thing that we really need to show is that the behavior of the formula $\psi_\sim$ is actually the one we described in Section 5, that is, we need to prove that $\psi_\sim$ forces each configuration to be the result of the application of a transition of a (generic) counter machine to the previous configuration. To this end, we show that each configuration τ_{i+1} contains an exact copy of the counter-points of the previous configuration τ_i in its initial part plus possibly an additional point labeled with *plus* at the end (if it is obtained from the previous configuration by an increment transition).

Let $\mathcal{S} = (\mathbb{I}(N), A, B, \sim_1, \sim_2, V)$ be a model of ψ_{M,q_0,q_f} and $q^0, q^1, \ldots, q^m$ be the enumeration of its state-points according to the order of the domain (therefore, by $\psi_{0\to 0}$, $q^0 = q_0$ and $q^m = q_f$). The configuration τ_i is defined as the set of points from q^i (included) to q^{i+1} (excluded). τ_m consists of q^m and the points that follows it, till the end of the domain $\mathbb{I}(N)$. We denote by $\sim_k$ any of the relations $\sim_1$ or $\sim_2$. If τ is an ordered set of points, we write τ^j to denote the j-th point of τ.

Let f be the set of all and only those pairs of points, belonging to two consecutive configurations τ_i and τ_{i+1}, which are connected by a linking interval, that is, $(x, y) \in f$ if and only if there exist τ_i, τ_{i+1} such that $x \in \tau_i$, $y \in \tau_{i+1}$, $x \sim_1 y$, and $x \sim_2 y$.

First of all, we observe that for each counter-point x in τ_i there exists a counter-point y in τ_{i+1} such that $(x, y) \in f$ (by (C4)). Moreover, by condition (C5), every pair in f consists of counter-points with the same label c_k. We now prove that f is (the map of) an injective function that preserves adjacency between points, that is, if x, x' are consecutive counter-points, then $f(x), f(x')$ are consecutive counter-points as well:

- f is a function: suppose that there exist y, y', with $y < y'$, in τ_{i+1} which have the same counter-image x, with $x \in \tau_i$, in f, then $y \sim_1 x \sim_1 y'$ and $y \sim_2 x \sim_2 y'$, and thus $y \sim_1 y'$ and $y \sim_2 y'$, which violates condition (C1) or condition (C2), depending on the distance between y and y' (if $|y' - y| = 1$, then it violates (C1); otherwise, it violates (C2));
- f is injective: the proof is similar to the one given for the previous point. Suppose that there exist x, x', with $x < x'$, in τ_i which have the same image y, with $y \in \tau_{i+1}$, in f, then $x \sim_1 y \sim_1 x'$ and $x \sim_2 y \sim_2 x'$, and thus $x \sim_1 x'$ and $x \sim_2 x'$, that violates condition (C1) or condition (C2);
- f preserves adjacency: let x, x' be two consecutive points in τ_i, that is, x' is the successor of x, and let y and y' be their respective images (since f is a injective function, it immediately follows that they are unique). By condition (C3), it holds that $\sim_k \in V([x, x'])$, and thus $y \sim_k x \sim_k x' \sim_k y'$ and $\sim_k \in V([y, y'])$, which, by condition (C2), implies that y, y' are consecutive.

By the properties of f and condition (C6), it follows that $f(\tau_i^j) = \tau_{i+1}^j$ for $j = 2, \ldots, |\tau_i|$ (order preservation). A graphical account of the relationships between pairs of counter-points belonging to two consecutive configurations is given in

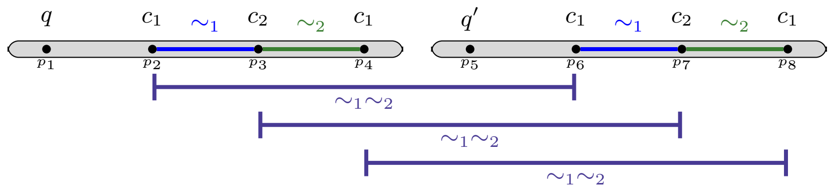


Fig. 2. Correspondence between counter-points of two consecutive configurations.

Figure 2. Moreover, thanks to condition (C7), the inequality $|\tau_i| \leq |\tau_{i+1}| \leq |\tau_i|+1$ holds, and thus the possible extra point of τ_{i+1} must have label *plus*. By this fact and the consistency of the labels *plus*, *minus*, and *del* guaranteed by condition (C5), it follows that τ_{i+1} is obtained from τ_i by an increment transition (if it has an extra point labeled with *plus*) or by a decrement transition (if it has no extra point), as desired.

It is worth emphasizing that the role of $\psi_\sim$ is not to guarantee that the transition applied to τ_i to obtain τ_{i+1} is a valid transition for M (which is the job of ψ_δ), but to ensure that τ_{i+1} *could* be obtained from τ_i by a transition of a counter machine.

7 Conclusions

In [15], Montanari and Sala studied complexity and expressiveness of the interval temporal logic AB$\sim$, that extends AB with an equivalence relation. Complexity and (un)decidability results are given by means of suitable reductions from reachability problems for lossy counter machines. The resulting picture is as follows: one gets decidability over finite linear orders with nonprimitive recursive complexity and undecidability over $\mathbb{N}$. In addition, they showed that decidability can be recovered by suitably restricting the class of models over which AB$\sim$ formulas are interpreted. In this paper, we proved that decidability of finite satisfiability is lost when two or more equivalence relations are added to AB.

As for future work, in analogy to what Montanari and Sala did for AB$\sim$ over the natural numbers, we are thinking of possible ways of restricting the class of models over which AB$\sim_1\sim_2$ formulas are interpreted in order to recover finite satisfiability. We are also studying the effects of the addition of two or more equivalence relations to other interval logics, such as (metric) propositional neighborhood logic, which preserves decidability when extended with one equivalence relation only [13].

References

1. J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. M. Bojańczyk. Weak MSO with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011.

3. M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.
4. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.
5. D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Interval temporal logics: a journey. *Bulletin of the EATCS*, 105:73–99, 2011.
6. S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.
7. J. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.
8. E. Kieronski, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. In *Proc. of the 27th LICS*, pages 431–440. IEEE, 2012.
9. E. Kieronski and M. Otto. Small substructures and decidability issues for first-order logic with two variables. In *Proc. of the 20th LICS*, pages 448–457. IEEE, 2005.
10. E. Kieronski and L. Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *Proc. of the 24th LICS*, pages 123–132. IEEE, 2009.
11. R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1-3):337–354, 2003.
12. M. L. Minsky. Computation: finite and infinite machines, 1967. *Cited on*, page 54, 1967.
13. A. Montanari, M. Pazzaglia, and P. Sala. Metric propositional neighborhood logic with an equivalence relation. In *Proc. of the 21st TIME*. IEEE, 2014.
14. A. Montanari, G. Puppis, P. Sala, and G. Sciavicco. Decidability of the interval temporal logic $AB\bar{B}$ on natural numbers. In *Proc. of the 27th STACS*, pages 597–608, 2010.
15. A. Montanari and P. Sala. Adding an equivalence relation to the interval logic $AB\bar{B}$: Complexity and expressiveness. In *Proc. of the 28th LICS*, pages 193–202. IEEE, 2013.
16. J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.
17. P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

Efficient channel assignment for cellular networks modeled as honeycomb grid

Soumen Nandi[1], Nitish Panigrahy[2], Mohit Agrawal[2],
Sasthi C. Ghosh[1], and Sandip Das[1]

[1] Advanced Computing and Microelectronics Unit,
Indian Statistical Institute, Kolkata, India,
{soumen.nandi_r, sasthi, sandipdas}@isical.ac.in.
[2] Department of Computer Science and Engineering,
National Institute of Technology, Rourkela, India,
{nitish.pani, mohitag25}@gmail.com.

Abstract. The channel assignment problem with separation is formulated as a vertex coloring problem of a graph $G = (V, E)$ where each vertex represents a base station and two vertices are connected by an edge if their corresponding base stations are interfering to each other. The $L(\delta_1, \delta_2, \cdots, \delta_t)$ coloring of G is a mapping $f : V \rightarrow \{0, 1, \cdots, \lambda\}$ such that $|f(u) - f(v)| \geq \delta_i$ if $d(u, v) = i$, where $d(u, v)$ denotes the distance between vertices u and v in G and $1 \leq i \leq t$. Here λ, the largest color assigned to a vertex of G, is known as the *span*. The same color can be reused in two vertices u and v if $d(u, v) \geq t+1$, where $t+1$ is the *reuse distance*. The objective is to minimize λ over all such coloring function f. Here $(\delta_1, \delta_2, \cdots, \delta_t)$ is called the *separation vector* where $\delta_1, \delta_2, \cdots, \delta_t$ are positive integers with $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_t$. Let λ^* be the minimum span such that there exists an $L(1, 1, \cdots, 1)$ coloring of G. We denote the separation vector $(1, 1, \cdots, 1)$ as (1^t). We deal with the problem of finding the maximum value of δ_1 such that there exists an $L(\delta_1, 1^{t-1})$ coloring with span equal to λ^*. So far bounds on δ_1 have been obtained for $L(\delta_1, 1^{t-1})$ coloring with span λ^* for the square and triangular grids. Shashanka et al. [18] posed the problem as open for the honeycomb grid. We give lower and upper bounds of δ_1 for $L(\delta_1, 1^{t-1})$ coloring with span λ^* of the honeycomb grid. The bounds are asymptotically tight. We also present color assignment algorithms to achieve the lower bound.

1 Introduction

In cellular networks, a large number of base stations is expected to cover a communication region. Such a covering can be achieved by placing the base stations according to a regular plane tessellation. It is well-known that only three different regular tessellations of the plane exist [6]. Specifically, the honeycomb, square and triangular tessellations cover the plane respectively by regular hexagons, squares, and triangles leading to three well-known topologies: honeycomb, square and triangular grids [6]. These three grid structures are shown in Figs. 1 (a), (b) and (c) where each vertex represents a base station and two vertices have an edge between them if their corresponding base stations are interfering to each other. Considering the network cost as a product of degree and diameter the honeycomb grid beats both the triangular and square grids as argued by Bertossi et al. [6]. The brick representation of the honeycomb grid has been shown in Fig. 1 (d) [6]. In this brick representation, the honeycomb grid can be viewed as a 2-dimensional grid. Thus each vertex can be represented by a 2-dimensional cartesian co-ordinate (i, j) where i and j are integers.

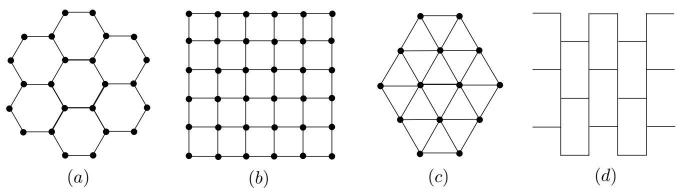


Fig. 1: (a) Honeycomb grid, (b) Square grid, (c) Triangular grid and (d) Brick structure of honeycomb grid.

The assignment of frequency channels to the base stations became a problem for enormous growth of wireless network. Since the number of available frequency channels is very limited, they must be utilized in an efficient manner. The main difficulty in efficient use of these frequency channels is the interference caused by unconstrained simultaneous transmissions of nearby stations. The same frequency channel can be reused by two stations provided that they are sufficiently far away so that the interference arisen between them can be negligible. However, the frequencies assigned to two nearby stations must differ by certain minimum value depending on the distance between them to avoid the channel interference. The channel assignment problem (CAP) deals with the task of assigning frequency channels to the stations such that there is no interference between the frequencies assigned to the nearby stations. The objective is to minimize the required span (bandwidth) where the span is represented by the difference between the least and the highest channel used. The minimum distance at which a channel can be reused with no interference is called the *reuse distance.*

The cellular network is often modeled as a graph $G = (V, E)$ where each vertex represents a base station and there is an edge between two vertices if their corresponding base stations are within the interference range of each other. Thus the channel assignment problem is basically a graph coloring problem on this graph. More formally, the $L(\delta_1, \delta_2, \cdots, \delta_t)$ coloring of a graph $G = (V, E)$ is a way to assign colors in $\{0, 1, \cdots, \lambda\}$ to the vertices of G using as small λ as possible such that the colors assigned to the vertices say u and v which are distance i apart differ by at least δ_i where $1 \leq i \leq t$ and the same color can be reused to two vertices if they are distance $t + 1$ or more apart [14]. Here $t + 1$ is the reuse distance, λ is the span and $(\delta_1, \delta_2, \cdots, \delta_t)$ is known as the separation vector where $\delta_1, \delta_2, \cdots, \delta_t$ are positive integers with $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_t$. Let λ^* be the minimum span required for any $L(1^t)$ coloring of G. It is evident that minimum span required for any $L(\delta_1, 1^{t-1})$ coloring of G must be greater than or equal to λ^* for any $\delta_1 > 1$. In this paper, we deal with the problem of finding the maximum value of δ_1 such that there exists an $L(\delta_1, 1^{t-1})$ coloring of G using the span equal to λ^*. Keeping the span restricted to the value of λ^* ensures that such an $L(\delta_1, 1^{t-1})$ coloring is always optimal. The rationale behind maximizing δ_1 is as follows. The interference between two adjacent channels can be prevented by using a guard band which is an unused part of the radio spectrum. Interference can also be prevented by using a specific channel separation requirement between two adjacent channels. It has been argued by Bertossi et al. [6] that when no extra colors are used, use of channel separation is always a better option than using guard bands between two adjacent channels. Moreover, higher channel separation between adjacent vertices will give the better quality of communication. So far bounds on δ_1 have been obtained for $L(\delta_1, 1^{t-1})$ coloring with span λ^* for square and triangular grids [5, 18]. Bertossi et al. [6] proposed an algorithm for optimal $L(1^t)$ coloring for the honeycomb grid. Shashanka et al. [18] posed $L(\delta_1, 1^{t-1})$ coloring of honeycomb grid for $\delta_1 > 1$ as an open problem. We found lower and upper bounds of δ_1 for $L(\delta_1, 1^{t-1})$ coloring with span λ^* of the honeycomb grid. We also present color assignment algorithms to achieve the lower bound. The obtained bounds are asymptotically tight.

This paper is arranged in the following way. In section 2, we have described some related works. In section 3, we have presented the basic concepts and notations. In section 4, we have provided the bounds of δ_1 in $L(\delta_1, 1^{t-1})$ coloring of the honeycomb grid. We have also given assignment algorithms to achieve the lower bound in this section. Section 5 concludes the paper.

2 Related work

The $L(1^t)$ coloring has been widely studied by several authors [2, 3, 10, 15, 16] for many special type of graphs. The intractability of optimal $L(1^t)$ coloring, for any positive integer t, has been proved by McCormick [15] for arbitrary graphs. The optimal $L(1^t)$ colorings for rings, square grids, and honeycomb grids have been proposed in [6, 3] and in [1] for trees and interval graphs. The optimal $L(\delta_1, 1^{t-1})$ colorings have been proposed in [5, 18] for rings, square grids, and cellular grids. The optimal $L(\delta_1, \delta_2)$ coloring on square and triangular grids have been proposed [11]. Chang et al. [9] gave bounds for $L(\delta_1, 1)$ coloring of chordal graphs and trees. Griggs and Jin [12, 13] provided optimal $L(\delta_1, 1)$ coloring for buses, rings, wheels, trees, and regular grids where δ_1 is a non-negative real number. The optimal $L(2, 1, 1)$ coloring for square grids [11] and triangular grids, honeycomb grids, and rings [4, 5] have been proposed. The $L(\delta_1, \delta_2, 1)$ coloring for squared and eight-regular grids has been studied in [8]. The $L(2, 1)$ coloring has been investigated in [7, 19] for different graphs.

All these results stated above basically deal with finding minimum span for the concerned coloring of different graphs. Our focus is, however, to find the maximum separation between two colors assigned to two adjacent vertices in an

$L(\delta_1, 1^{t-1})$ coloring of the honeycomb grid. Though the problem has been solved for the triangular and square grids, it was open for the honeycomb grid [18].

3 Basic Concepts and Notations

The required span for $L(\delta_1, 1^{t-1})$ coloring will definitely be greater or equal to the required span for $L(1^t)$ coloring. We now define the *Distance-t clique* of a graph G in order to establish bounds on the required minimum span for $L(1^t)$ coloring of G.

Definition 1. *The Distance-t clique of a graph $G = (V, E)$ is an induced subgraph $G' = (V', E')$ where distance between every pair of vertices in V' is at most t. A maximum Distance-k clique is the Distance-k clique where cardinality of V' is maximum [17]. We denote a maximum Distance-t clique of a graph by D_t.*

The standard graph theoretic term, maximum clique, is a Distance-t clique with $t = 1$. So, Distance-t clique of a graph $G = (V, E)$ is a subgraph of G with *diameter t*. Though finding D_t for general graph is a hard problem, it can be found for the honeycomb grid [6]. As for example, D_1 with 2 vertices, D_3 with 6 vertices, D_5 with 14 vertices, D_0 with 1 vertex, D_2 with 4 vertices and D_4 with 10 vertices of the honeycomb grid are shown in Figs. 2 (a), (b), (c), (d), (e) and (f) respectively.

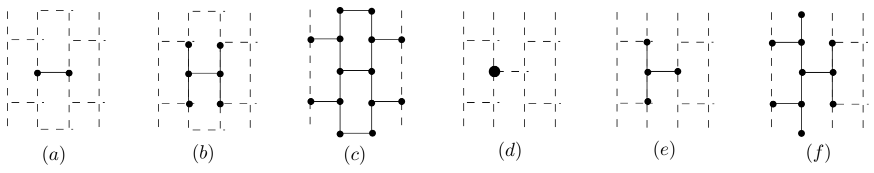


Fig. 2: (a) D_1, (b) D_3, (c) D_5, (d) D_0, (e) D_2 and (f) D_4.

The required span for any $L(1^t)$ coloring of G will be at least the cardinality of D_t. Our objective is to find the maximum value of δ_1 such that there exists an $L(\delta_1, 1^{t-1})$ coloring of G using the set of colors from $\{0, 1, \cdots, \lambda - 1\}$, where λ is the cardinality of D_t. We denote δ_1^{max} as the maximum value of δ_1. Note that δ_1 represents the minimum frequency separation requirement between any two adjacent vertices. In [6], the minimum λ for $L(1^t)$ coloring of D_t in honeycomb grid was computed by considering 8 cases with $t = 8p+q$ where $0 \leq q \leq 7$. Based on the results reported in [6], we can state the following result by considering only 4 cases with $t = 4p + q$ where $0 \leq q \leq 3$.

Result 1 *In honeycomb grid, the minimum λ for any $L(1^t)$ coloring of D_t can be computed as:*

$$\lambda \geq \begin{cases} 6p^2 + 6p + 2, & \text{if } t = 4p + 1 \\ 6p^2 + 12p + 6, & \text{if } t = 4p + 3 \\ 6p^2 + 3p + 1, & \text{if } t = 4p \\ 6p^2 + 9p + 4, & \text{if } t = 4p + 2, \end{cases}$$

where p is a non negative integer.

4 $L(\delta_1^{max}, 1^{t-1})$ coloring of the honeycomb grid

4.1 Upper bound of δ_1^{max}

Let us consider $p = 1$ and $t = 4 \times 1 + 1 = 5$. By putting these values in Result 1, minimum λ is found to be 14. The subgraph D_5 constituted by these 14 vertices is shown in Fig. 2 (c). Observe that in D_5 (Fig. 2 (c)) all the cycles are of even length, i.e., it is a bipartite graph. We see that there are 7 disjoint edges. For a bipartite graph like this each partition contains 7 nodes. We can select seven nodes of one partition and assign colors 0 to 6 to them and for the remaining seven nodes from the other partition, we can assign colors 7 to 13 as shown in Fig. 3. The δ_1^{max} obtained by this assignment is found to be 7. We now state a bound on δ_1^{max} in the following Lemma 1.

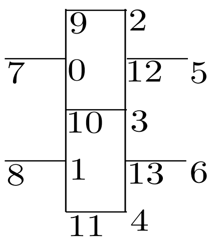


Fig. 3: A coloring of D_5 where $\delta_1^{max} = \frac{\lambda}{2}$.

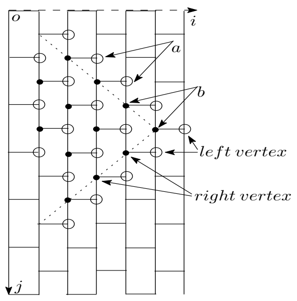


Fig. 4: 18 nodes of label a which are neighbors of 12 nodes of label b.

Lemma 1. *For any $L(\delta_1^{max}, 1^{t-1})$ coloring of D_t of a honeycomb grid, $\delta_1^{max} \leq \frac{\lambda}{2}$, where λ is the cardinality of D_t.*

Proof. For λ number of nodes there can be at most $\frac{\lambda}{2}$ number of disjoint edges in D_t. So we can form a maximum independent set of length at most $\frac{\lambda}{2}$. And our δ_1^{max} therefore, can be at most $\frac{\lambda}{2}$. Thus we can conclude that $\delta_1^{max} \leq \frac{\lambda}{2}$. We can compute λ for D_t by applying Result 1. $\square$

Remark 1. The above result considers the assignment of colors to D_t only. However, as practical networks are very large, we have to repeat the assignment of this subgraph in a regular fashion so as to cover the entire honeycomb grid. If we want to repeat an assignment of D_t in a regular fashion with a view to covering the entire grid, we will see that $\delta_1^{max} = \frac{\lambda}{2}$ may not be achievable. Consider the assignment of D_5 as shown in Fig. 3. It can easily be verified that with this assignment, $\delta_1^{max} = \frac{\lambda}{2} = 7$. Note that here we have considered the assignment of D_5 only but not considered the possibility of repeating this assignment to cover the entire grid. We now consider two different assignments of D_5 and their repetition pattern to cover the entire grid. As for example one of them is shown in Fig. 5 (a). With this repetition pattern, $\delta_1^{max} \neq 7$, rather $\delta_1^{max} = 2$. The other one is shown in Fig. 5 (b), where $\delta_1^{max} = 4$ is achieved. Our objective is to find an assignment of the entire grid for which the value of δ_1^{max} is maximized.

As far as the assignment of D_t is concerned, there are three types of vertices with degrees 1, 2 and 3. Here the degree of a vertex is computed based on the number of adjacent vertices which are already assigned within the subgraph. If we consider the repetition of the assignment of this subgraph to cover the entire grid, the degree of each vertex will eventually become 3. Moreover, the assignment of D_t should be repeated to infinity in such a way that the colors of the adjacent three vertices of a vertex with a particular color remain fixed throughout the entire grid. Such assignment is possible which is given in Fig. 5 (b). As for example, the vertex of color 3 is having the vertices of colors 7, 8 and 10 as its adjacent everywhere. We now have the following result on δ_1^{max} that considers the assignment of the entire honeycomb grid.

Lemma 2. *In $L(\delta_1^{max}, 1^{t-1})$ coloring of honeycomb grid $\delta_1^{max} \leq \frac{\lambda}{2} - 1$, where λ is the cardinality of D_t.*

Proof. From Lemma 1, it follows that the maximum value for δ_1^{max} without repetition is $\frac{\lambda}{2}$. So with repetition, δ_1^{max} cannot exceed this value due to additional constraints on the vertices of degree not equal to 3 which have now become vertices of degree 3 each. Now consider the vertex colored as $\delta_1^{max} - 1$. Because of the color separation δ_1^{max} between two adjacent vertices, the neighbors of the vertex colored with $\delta_1^{max} - 1$, must have a color larger than $\delta_1^{max} - 1$. So its adjacent vertices are of colors with a difference of at least δ_1^{max} which are $2\delta_1^{max} - 1$, $2\delta_1^{max}$ and $2\delta_1^{max} + 1$. But a vertex can have a color at most $\lambda - 1$. So $2\delta_1^{max} + 1 \leq \lambda - 1$, i.e., $\delta_1^{max} \leq \frac{\lambda}{2} - 1$. □

Remark 2. The honeycomb grid is a bipartite graph. We label the vertices of one partition as a and that of other partition as b. The basic idea behind the result stated in Lemma 2 is that there are exactly 3 vertices of label a which are adjacent to a vertex of label b. We observe that there are a minimum of 5 vertices of label a which are adjacent to two vertices of label b. This is because if we choose any two vertices of label b there can be at most one common adjacent (neighbor) vertex of those two b labeled vertices. With a view to generalizing this idea, we required to find the minimum number of vertices of label a which are adjacent to n vertices of label b. Consider $l = 1+\max\ \{i \mid \sum_{j=1}^{i} j < n\}$.

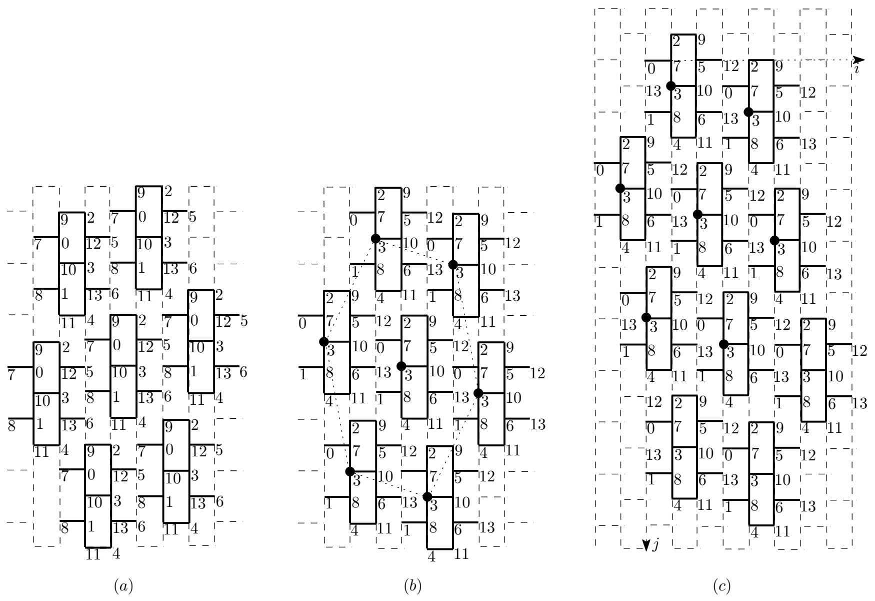


Fig. 5: (a) Coloring of D_5 with repetition where $\delta_1^{max} \leq \frac{\lambda}{2}$ is not achieved, (b) Infinite repetition of D_5 in a honeycomb grid where reuse distance is satisfied and (c) Coloring of D_5 with repetition where $\delta_1^{max} = 4$.

Observe that l depends on n and can be expressed as $l = \lceil \frac{\sqrt{8n+1}-1}{2} \rceil$. We now have the following result.

Lemma 3. *The minimum number of neighbors of n nodes of label b in a honeycomb grid is $n + l + 1$, where no two b labeled vertices are adjacent.*

Proof. Let us consider the honeycomb grid which is shown in Fig. 4. Consider l consecutive columns in that graph. Assume that the number of b labeled vertices in ith column is k_i, $1 \leq i \leq l$, where no two b labeled vertices are adjacent. So, $\sum_{i=1}^{l} k_i = n$. Note that each b labeled vertex has two neighbors within the same column and one neighbor in the $(i+1)$th column. We denote the two neighbors of a b labeled vertex within the same column as the *same column neighbors* and the neighbor in the adjacent column $((i+1))$ as the *adjacent column neighbor*. Observe that the number of same column neighbors of the b labeled vertices of ith column is at least $(k_i + 1)$ and at most $2k_i$. The number of adjacent column neighbors of the b labeled vertices of ith column is exactly k_i. Note, however, that the adjacent column neighbors of the b labeled vertices of ith column may share some same column neighbors of the b labeled vertices of $(i+1)$th column. Observe that the least number of same column neighbors of the b labeled vertices of ith column is $(k_i + 1)$ when all the b labeled vertices are placed one after another at two distance apart. So the total number of same column neighbors of the b labeled vertices across all the l consecutive columns is $\sum_{i=1}^{l}(k_i + 1)$. Note that the k_i number of adjacent column neighbors of the b labeled vertices of ith column may equal or less or greater than the $k_{i+1}+1$ number of same column neighbors of the b labeled vertices of $(i+1)$th column. Let N_i be the number of adjacent column neighbors of the b labeled vertices of ith column. So the total number of adjacent column neighbors of the b labeled vertices across all the l consecutive columns is $\sum_{i=1}^{l} N_i$, where

$$N_i = \begin{cases} k_i, & \text{if } i = l \\ k_i - (k_{i+1}+1), & \text{if } k_i > k_{i+1}+1 \text{ and } 1 \le i \le l-1 \\ 0, & \text{if } k_i \le k_{i+1}+1 \text{ and } 1 \le i \le l-1. \end{cases}$$

So, the total number of neighbors of b labeled vertices across all the l consecutive columns is $\sum_{i=1}^{l}(k_i+1) + \sum_{i=1}^{l} N_i$. If $k_i < k_{i+1}+1$, for all i, then the value of k_l becomes largest and if $k_i > k_{i+1}+1$, for all i, then the total number of adjacent column neighbors will be increased. As our objective is to find the minimum number of neighbors, the best possible situation is

$$k_i = \begin{cases} 1, & \text{if } i = l \\ k_{i+1}+1, & \text{if } 1 \le i \le l-1. \end{cases}$$

So, the minimum value of the above expression is $\sum_{i=1}^{l}(k_i+1) + \sum_{i=1}^{l} N_i = n + l + 1$. We conclude that the minimum number of neighbors of n nodes of b labeled vertices in a honeycomb grid is $n + l + 1$. $\square$

Example 1. Corresponding to Lemma 3, we are presenting one example for $n = 12$ in Fig. 4, where 12 can be expressed as $2+4+3+2+1$. Here all the 12 nodes of label b are shown by dots and nodes of label a are circled which are the neighbors of b labeled vertices. We see that minimum number of neighbors of 12 nodes of label b is $12+5+1 = 18$.

Theorem 1. *In $L(\delta_1^{max}, 1^{t-1})$ coloring of honeycomb grid, $\delta_1^{max} \le \frac{\lambda}{2} - \lceil \frac{\sqrt{8\lfloor \frac{\lambda}{4} \rfloor + 1} - 1}{2} \rceil - 1$, where λ is the cardinality of D_t.*

Proof. We know that any consecutive δ_1^{max} non negative integers will be forming an independent set. Let us consider such an independent set $S = \{a_1, a_2, \cdots, a_{\delta_1^{max}}\}$ such that all its elements are consecutive and in increasing order. As D_t is a bipartite graph with cardinality λ and λ is even, we can label $\frac{\lambda}{2}$ vertices of it by a label say a and the remaining $\frac{\lambda}{2}$ vertices by an another label say b. Let us denote the vertices colored by $0, 1, \cdots, \frac{\lambda}{2}-1$ as label a vertices and the vertices colored by $\frac{\lambda}{2}, \frac{\lambda}{2}+1, \cdots, \lambda-1$ as label b vertices. We can easily construct an independent set S with $\delta_1^{max} - 2$ vertices of label a and 2 vertices of label b. Essentially S contains the a labeled vertices colored by $\frac{\lambda}{2} - (\delta_1^{max}-2), \frac{\lambda}{2} - (\delta_1^{max}-3), \cdots, \frac{\lambda}{2}-1$ and the b labeled vertices colored by $\frac{\lambda}{2}$ and $\frac{\lambda}{2}+1$. Now to form an independent set any one of the δ_1^{max} vertices of set S must not be adjacent to other vertex of S. At minimum there can be 5 vertices which will be adjacent to these 2 vertices of label b. To form an independent set these 5 nodes should not be in the set S. So to choose $\delta_1^{max} - 2$ vertices of label a in the set S we are left with $\frac{\lambda}{2} - 5$ number of choices of a labeled vertices. Hence $\delta_1^{max} - 2 \leq \frac{\lambda}{2} - 5$ i.e. $\delta_1^{max} \leq \frac{\lambda}{2} - 3$.

Similarly, we can construct an independent set S with $\delta_1^{max} - \lfloor \frac{\lambda}{4} \rfloor$ vertices of label a and $\lfloor \frac{\lambda}{4} \rfloor$ vertices of label b. If we take the number of vertices of label b more than $\lfloor \frac{\lambda}{4} \rfloor$, then we can think of switching the labels, i.e., all the a labeled vertices will be changed to label b and all the b labeled vertices will be changed to label a so that same case will again be happened. So $\lfloor \frac{\lambda}{4} \rfloor$ is the threshold value. Hence, by Lemma 3, the minimum number of neighbors of $\lfloor \frac{\lambda}{4} \rfloor$ vertices of label b is $\lfloor \frac{\lambda}{4} \rfloor + \lceil \frac{\sqrt{8\lfloor \frac{\lambda}{4} \rfloor + 1} - 1}{2} \rceil + 1$. Therefore, to choose $\delta_1^{max} - \lfloor \frac{\lambda}{4} \rfloor$ vertices of label a in the set S we are left with $\frac{\lambda}{2} - [\lfloor \frac{\lambda}{4} \rfloor + \lceil \frac{\sqrt{8\lfloor \frac{\lambda}{4} \rfloor + 1} - 1}{2} \rceil + 1]$ vertices of label a. So $\delta_1^{max} - \lfloor \frac{\lambda}{4} \rfloor \leq \frac{\lambda}{2} - [\lfloor \frac{\lambda}{4} \rfloor + \lceil \frac{\sqrt{8\lfloor \frac{\lambda}{4} \rfloor + 1} - 1}{2} \rceil + 1]$. Hence the result. □

4.2 Lower bound of δ_1^{max}

As mentioned earlier, the honeycomb grid can be viewed as a 2-dimensional grid where each vertex can be represented as (i, j) for some integers i and j. We call a vertex (i, j) as a *right vertex* if its adjacent 3 vertices are $(i, j + 1)$, $(i, j - 1)$ and $(i + 1, j)$. Similarly a vertex (i, j) is called a *left vertex* if its adjacent 3 vertices are $(i, j + 1)$, $(i, j - 1)$ and $(i - 1, j)$. Note that a left vertex has a neighbor at left side and a right vertex has a neighbor at right side. In Fig. 4, the vertices marked by circle and dots are left and right vertices respectively. It is easy to verify that all the left and right vertices are forming two separate independent sets. So we can label all the left vertices by a and all the right vertices by b. Let us consider the assignment of the first $\frac{\lambda}{2}$ colors $0, 1, \cdots, \frac{\lambda}{2} - 1$ to the b labeled vertices and the rest $\frac{\lambda}{2}$ colors $\frac{\lambda}{2}, \frac{\lambda}{2} + 1, \cdots, \lambda - 1$ to the a labeled vertices in some fashion. We now describe an assignment scheme for $L(\delta_1^{max}, 1^{t-1})$ coloring of the honeycomb grid, where t is odd and p is any non negative integer.

$L(\delta_1^{max}, 1^{t-1})$ coloring of a honeycomb grid for odd t: We first deal with the $L(\delta_1^{max}, 1^{t-1})$ coloring algorithm for the case of $t = 4p + 1$. As our objective is to maximize the value of δ_1^{max}, we assign the colors ranging from 0 to $\frac{\lambda}{2} - 1$ to the right vertices of D_{4p+1} sequentially starting from the left most column, top to bottom, towards the 2nd right most column. Similarly, the colors ranging from $\frac{\lambda}{2}$ to $\lambda - 1$ are assigned to the left vertices of D_{4p+1} sequentially starting from the 2nd left most column, top to bottom, towards the right most column. Though there are $(2p + 2)$ columns in D_{4p+1} only the first $(2p + 1)$ columns contain vertices of label b. In each column from left most to 2nd right most there are $(p + 1)$, $(p + 2)$, $\cdots$, $2p$, $(2p + 1)$, $2p$, $\cdots$, $(p + 2)$, $(p + 1)$ number of b labeled vertices respectively. So, the number of colors in column i is denoted by

t_i and can be defined by

$$t_i = \begin{cases} p+i, & \text{if } 1 \leq i \leq p \\ 2p+1, & \text{if } i = (p+1) \\ 3p+2-i, & \text{if } (p+2) \leq i \leq (2p+1). \end{cases}$$

We see that $(t_1 + t_{p+2}) = (t_2 + t_{p+3}) = \cdots = (t_p + t_{2p+1}) = (3p+1)$. An example of execution of this algorithm to the subgraph D_{4p+1} is shown in Fig. 5 (c) for the case where $p = 1$. So far we have considered the color assignment of the vertices of D_{4p+1} only. We now consider the repetition of these colors to cover the entire grid. We use the following repetition pattern to extend the coloring of D_{4p+1} to the entire grid: a color assigned to vertex (i, j) is repeated to exactly six vertices $(i+p+1, j+3p+1)$, $(i+2p+1, j-1)$, $(i+p, j-3p-2)$, $(i-p-1, j-3p-1)$, $(i-2p-1, j+1)$ and $(i-p, j+3p+2)$ forming a hexagon of radius $t+1$ (reuse distance) centered around the vertex (i, j). In Fig. 5 (c), the repetition of color 3 has been explicitly shown by filled circle. The same repetition pattern holds for each color assigned to the vertices of D_{4p+1}. It is evident that this pattern can be repeated infinitely and it satisfies the reuse distance. Using this pattern of repetition, when the assignment of D_{4p+1} is repeated infinitely, we observe that all $\sum_{i=1}^{2p+1} t_i$ colors are placed in each column. The sequence of colors used in each column can be described as follows: The sequence is actually composed of $(2p+1)$ subsequences where kth subsequence starts with color f_k and ends with color $f_k + T_k - 1$, where $1 \leq k \leq 2p+1$. That means, there are T_k many colors in the kth subsequence. Let us define $T_k = t_{\frac{k+1}{2}}$ when k is odd and $T_k = t_{p+\frac{k+2}{2}}$ when k is even. We now define f_k as follows: $f_1 = 0$, $f_2 = \sum_{i=1}^{p+1} t_i = \frac{3p^2+5p+2}{2}$ and in general,

$$f_k = \begin{cases} \sum_{i=0}^{\frac{k-3}{2}} T_{2i+1}, & \text{if } 2p+1 \geq k > 2 \text{ and k is odd} \\ f_k = \frac{3p^2+5p+2}{2} + \sum_{i=1}^{\frac{k-2}{2}} T_{2i}, & \text{if } 2p+1 \geq k > 2 \text{ and k is even.} \end{cases}$$

Thus the sequence of colors used in each column is as follows: $f_1, f_1 + 1, \cdots, f_1 + T_1 - 1$; $f_2, f_2 + 1, \cdots, f_2 + T_2 - 1$; $\cdots$; $f_k, f_k + 1, \cdots, f_k + T_k - 1$; $\cdots$; $f_{2p+1}, f_{2p+1} + 1, \cdots, f_{2p+1} + T_{2p+1} - 1$. If color f_1 is placed on (i, j), then on that same ith column f_1 is again placed at $(i, j + 6p^2 + 6p + 2)$ and on $(i+1)$th column f_1 is placed at $(i+1, j-(6p+3))$. The same pattern holds for all other colors. The coloring of D_5 and its repetition to cover the entire grid has been shown in Fig. 5 (c). In this case, there are 3 subsequences $0, 1$; $5, 6$; and $2, 3, 4$. So, the sequence of colors used in each column are $0, 1, 5, 6, 2, 3, 4$.

So far we have considered the assignment of label b vertices only. We now consider the assignment of a labeled vertices ranging from $\frac{\lambda}{2}$ to $\lambda - 1$. Let color $x \in [0, \frac{\lambda}{2} - 1]$ has been assigned to vertex (i, j). Then color $x + \frac{\lambda}{2} \in [\frac{\lambda}{2}, \lambda - 1]$ will be assigned to vertex $(i+1, j)$. It can now be seen that the infinite honeycomb grid will be filled up by λ colors ranging from 0 to $\lambda - 1$ where $\lambda = 6p^2 + 6p + 2$ and $t = 4p+1$. We observe that this assignment is a *no-hole* assignment meaning every colors ranging from 0 to $\lambda - 1$ has been used. An example of execution of the algorithm to the entire grid for $t = 4p + 1$ is shown in Fig. 5 (c) for the case where $p = 1$. Similarly the coloring scheme for the case of $t = 4p + 3$ can be obtained, which we omitted here due to space restriction.

Theorem 2. *For $L(\delta_1^{max}, 1^{t-1})$ coloring of the honeycomb grid,*

$$\delta_1^{max} \geq \begin{cases} 3p^2 + p, & \text{if } t = 4p + 1 \qquad (1) \\ 3p^2 + 3p + 1, & \text{if } t = 4p + 3 \qquad (2) \end{cases}$$

where λ is the cardinality of D_t and p is a non-negative integer.

Proof. Consider D_t with odd t. Depending on the value of t there are 2 cases.

Case 1. $t = 4p + 1$**:** We observe that 1st b labeled vertex in D_{4p+1} from the top of pth column is given by $\sum_{i=1}^{p-1} t_i = \frac{3p^2-3p}{2}$. The 1st and 2nd b labeled vertices from the top of $(p+1)$th column are $\sum_{i=1}^{p} t_i = \frac{3p^2+p}{2}$ and $\sum_{i=1}^{p} t_i + 1 = \frac{3p^2+p+2}{2}$ respectively. The 1st a labeled vertex from the top of $(p+1)$th column is $\frac{\lambda}{2} + \frac{3p^2-3p}{2}$, which is adjacent to all three said b labeled vertices. The color gap between the 2nd b labeled vertex of $(p+1)$th column and 1st a labeled vertex of $(p+1)$th column is minimum which is found to be:

$$\delta_1^{max} = \frac{\lambda}{2} + \frac{3p^2 - 3p}{2} - \frac{3p^2 + p + 2}{2} = 3p^2 + p.$$

Case 2. $t = 4p+3$**:** We observe that 1st b labeled vertex in D_{4p+3} of $(p+2)$th column is $\sum_{i=1}^{p+1} t_i = \frac{3p^2+7p+4}{2}$. The 1st a labeled vertex of $(p+1)$th column is $\frac{\lambda}{2} + \sum_{i=1}^{p}(p+i) = \frac{\lambda}{2} + \frac{3p^2+p}{2}$. The color gap between these two adjacent colors is minimum which is found to be:

$$\delta_1^{max} = \frac{\lambda}{2} + \frac{3p^2 + p}{2} - \frac{3p^2 + 7p + 4}{2} = 3p^2 + 3p + 1.$$

□

Observation 1 *For any $L(\delta_1^{max}, 1^{t-1})$ coloring of a honeycomb grid, the lower bound of δ_1^{max} obtained from Theorem 2 and the upper bound of δ_1^{max} obtained from Theorem 1 are asymptotically equal.*

Proof. Observe that the bound of δ_1^{max} obtained from Theorem 1 can be expressed as $3p^2 + cp + d$ where $1.268 \leq c \leq 4.268$ and d is a constant. Hence the result. □

***$L(\delta_1^{max}, 1^{t-1})$* coloring of a honeycomb grid for even *t*:** It follows from Lemmas 2 and 6 of [6] that the $L(\delta_1^{max}, 1^{t-1})$ coloring of D_t for any even t can not be repeated to cover the entire grid using the colors from 0 to $\lambda - 1$, where λ is the cardinality of D_t. It is observed that D_t with even t can be considered as a subgraph of D_{t+1} where $t+1$ is odd. Note that difference between the cardinality of D_{t+1} and D_t is $3p + 1$ when $t = 4p$ and $3p + 2$ when $t = 4p + 2$. Thus by putting these minimum number of extra colors, we can cover the entire grid for the case of even t. So the lower and upper bounds of δ_1^{max} is same as that of the case of odd t. Hence we can conclude the following theorem.

Theorem 3. *For $L(\delta_1^{max}, 1^{t-1})$ coloring of the honeycomb grid,*

$$\delta_1^{max} \geq \begin{cases} 3p^2 + p, & \text{if } t = 4p \quad (3) \\ 3p^2 + 3p + 1, & \text{if } t = 4p + 2 \quad (4) \end{cases}$$

where λ is the cardinality of D_t and p is a non-negative integer.

5 Conclusion

We have derived upper and lower bounds of δ_1^{max} for any $L(\delta_1^{max}, 1^{t-1})$ coloring with span λ^* of honeycomb grid, where λ^* is the minimum span required for any $L(1^t)$ coloring. We have shown that the bounds are asymptotically tight. We have also given assignment algorithms for finding the lower bounds of δ_1^{max}.

References

1. G. Agnarsson, R. Greenlaw, and M. Halldorson. On powers of chordal graphs and their colorings. *Congressus Numerantium*, 144:41–65, 2000.
2. R. Battiti, A. Bertossi, and M. Bonuccelli. Assigning codes in wireless networks: Bounds and scaling properties. *Wireless Networks*, 5:195–209, 1999.
3. A. Bertossi and M. Pinotti. Mappings for conflict-free access of paths in bidimensional arrays, circular lists, and complete trees. *Journal of Parallel and Distributed Computing*, 62:1314–1333, 2002.
4. A. Bertossi, M. Pinotti, and R. Tan. Efficient use of radio spectrum in wireless networks with channel separation between close stations. *DIAL M for Mobility: Intl ACM Workshop on Discrete Algorithms and Methods for Mobile Computing*, Boston, 2000.
5. A. Bertossi, M. Pinotti, and R. Tan. Channel assignment with separation for interference avoidance in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 14:222–235, 2003.
6. A. A. Bertossi, C. M. Pinotti, R. Rizzi, and A. M. Shende. Channel assignment for interference avoidance in honeycomb wireless networks. *Journal of Parallel and Distributed Computing*, 64:1329–1344, 2004.
7. T. Calamoneri. The l(h, k)-labelling problem: An updated survey and annotated bibliography. *Comput. J.*, 54(8):1344–1371, 2011.
8. T. Calamoneri. Optimal $l(\delta_1, \delta_2, 1)$-labeling of eight-regular grids. *Information Processing Letters*, 113:361–364, 2013.
9. G. J. Chang, W. T. Ke, D. Kuo, and R. K. Yeh. $l(d, 1)$-labeling of graphs. *Discrete Math.*, 220:57–66, 2000.
10. I. Chlamtac and S. Pinter. Distributed nodes organizations algorithm for channel access in a multihop dynamic radio network. *IEEE Transactions on Computers*, 36:728–737, 1987.
11. J. V. den Heuvel, R. A. Leese, and M. Shepherd. Graph labelling and radio channel assignment. *Journal of Graph Theory*, 29:263–283, 1998.

12. J. R. Griggs and X. T. Jin. Optimal channel assignments for lattices with conditions at distance two. *5th IEEE International Parallel and Distributed Processing Symposium, extended abstruct*, 2005.
13. J. R. Griggs and X. T. Jin. Real number graph labellings with distance conditions. *SIAM J. Discrete Math.*, 20:302–327, 2006.
14. J. R. Griggs and R. K. Yeh. Labelling graphs with a condition at distance 2. *SIAM J. Discrete Math.*, 5:586–595, 1992.
15. S. McCormick. Optimal approximation of sparse hessians and its equivalence to a graph coloring problem. *Mathematical Programming*, 26:153171, 1983.
16. A. Sen, T. Roxborough, and S. Medidi. Upper and lower bounds of a class of channel assignment problems in cellular networks. *Proc. of the INFOCOM*, 3:1284–1291, 1998.
17. A. Sen, T. Roxborough, and B. P. Sinha. On an optimal algorithm for channel assignment in cellular network. *Proc. of IEEE ICC*, pages 1147–1151, 1999.
18. M. V. S. Shashanka, A. Pati, and A. M. Shende. A characterisation of optimal channel assignments for cellular and square grid wireless networks. *Mobile Networks and Applications*, 10:89–98, 2005.
19. R. Yeh. A survey on labeling graphs with a condition at distance two. *Discrete Mathematics*, 306(12):1217–1231, 2006.

Programmable enforcement framework of information flow policies

Minh Ngo and Fabio Massacci

University of Trento, Italy
{surname}@disi.unitn.it

Abstract. We propose a programmable framework that can be easily instantiated to enforce a large variety of information flow properties. Our framework is based on the idea of secure multi-execution in which multiple instances of the controlled program are executed in parallel. The information flow property of choice can be obtained by simply implementing programs that control parallel executions. We present the architecture of the enforcement mechanism and its instantiations for non-interference (NI) (from Devriese and Piessens), non-deducibility (ND) (from Sutherland) and some properties proposed by Mantel, such as removal of inputs (RI) and deletion of inputs (DI), and demonstrate formally soundness and precision of enforcement for these properties.

Keywords: Non-Interference, Non-Deducibility, Possibilistic Information Flow Properties, Programming Language, Secure Multi Execution

1 Introduction

Computing systems often process data classified as *sensitive*, or, secret. To ensure security, treatment of these data has to comply with designated information flow policies that regulate whether the publicly visible behavior of a system can be influenced by secret data.

Non-interference (NI) [7] totally prevents leakage of secrets to public channels by requiring that the confidential information does not interfere with all events at the public levels. With or without the confidential information, observations at the public levels are still the same. By weakening or strengthening the definition of NI, security researchers have proposed different information flow properties. For example, declassification policies accept the behaviors in which some selected secret data can be released [14]. Sutherland defines [15] *non-deducibility* (ND), a stronger property than NI [6]. It assumes that an attacker knows the design of the observed program, and has partial access to the public program interfaces, and tries to infer the occurrence and non-occurrence of sequences of high input events. ND prevents the attacker from deducing which confidential event sequences have occurred or not.

Existing mechanisms for information flow policies enforcement and secure information release are based on several techniques: e.g., type systems [13], symbolic execution [2], multi-execution [5, 11], faceted values [1], etc. Yet these all

Table 1. EMs for the selected information flow policies

Policy	Components		
	MAP	REDUCE	T_M/T_R
Termination (in)sensitive non-interference [5]	Fig. 3a	Fig. 3b	Fig. 3
Termination (in)sensitive non-deducibility [15]	Fig. 3a	Fig. 3b	Fig. 7
Removal of inputs [9]	Fig. 8a	Fig. 3b	Fig. 8
Deletion of inputs [9]	Fig. 9a	Fig. 3b	Fig. 9

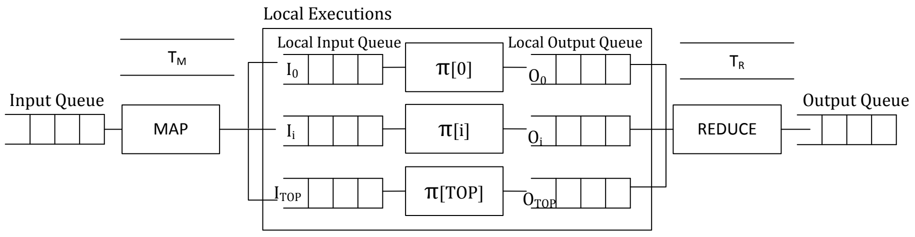


Fig. 1. Architecture of enforcement mechanisms

fall short in the same aspect: *these approaches work only for a single information flow policy*, typically NI or NI equipped with declassification. Modification of these mechanisms to enforce another information flow policy (for example, ND) is not straight-forward. Moreover, no run-time enforcement mechanism is proposed for ND.

1.1 The contribution of this paper

We propose a *programmable enforcement mechanism* (EM) that can be easily configured to enforce NI, ND and other information flow policies. Configurations of the EMs are summarized in Tab. 1. Our proposal is the first run-time EM that covers ND. SME by Devriese and Piessens [5] is a special case. Our EM relies on the secure multi-execution technique (SME) [5] in which multiple instances of the controlled program are executed in parallel and their input and output behaviors are controlled. To this construction we add two programmable components that map each input to the multiple instances and reduce output of the instances to a single output. We demonstrate soundness and precision of the proposed mechanisms using the operational semantics.

The rest of the paper is organized as follows. §2 gives an overview of the idea behind our approach and the architecture of the enforcement framework. Selected information flow policies and implementations of their EMs are introduced respectively in §3 and §4. Semantics of controlled programs and framework is introduced in §5. The soundness and precision of the EMs constructed are presented in §6. We discuss related work and conclude in §7.

2 Overview

Fig. 1 depicts the general architecture of the EM for an information flow property on a program π. It is composed by the MAP and REDUCE components, a stack EX of local executions ($\pi[0], \ldots,$ $\pi[TOP]$, where TOP is the index of the top of the stack), global input and output queues, and the tables T_M and T_R. Instructions used to compose controlled programs, MAP, and REDUCE programs are in respectively Fig. 2a, Fig. 2b, and Fig. 2c.

$$\begin{array}{lll}
\pi ::= & & \textit{program instructions}: \\
 & |x := e & \textit{assignment} \\
 & |\pi; \pi & \textit{sequence} \\
 & |\textbf{if } e \textbf{ then } \pi \textbf{ else } \pi & \textit{if} \\
 & |\textbf{while } e \textbf{ do } \pi & \textit{while} \\
 & |\textbf{skip} & \textit{skip} \\
 & |\textbf{input } x \textbf{ from } c & \textit{input} \\
 & |\textbf{output } e \textbf{ to } c & \textit{output}
\end{array}$$

(a) Basic instructions

$$\begin{array}{lll}
\pi_M ::= \pi & & \textit{instructions}: \\
 & |\textbf{map}(e, c, PRED[\,]) & \textit{map} \\
 & |\textbf{wake}(PRED[\,]) & \textit{wake} \\
 & |\textbf{clone}(PRED[\,], PRIV_{T_M}, PRIV_{T_R}) & \textit{clone}
\end{array}$$

(b) MAP instructions

$$\begin{array}{lll}
\pi_R ::= \pi & & \textit{instructions}: \\
 & |\textbf{retrieve } x \textbf{ from } (i, c) & \textit{retrieve} \\
 & |\textbf{wake}(PRED[\,]) & \textit{wake} \\
 & |\textbf{clean}(c, PRED[\,]) & \textit{clean}
\end{array}$$

(c) REDUCE instructions

π, e, x, and c are meta-variables for respectively instructions, expressions, variables, and input/output channels. A (controlled, MAP, or REDUCE) program is a sequence of instructions.

Fig. 2. Language instructions

Local executions are instances of the original program, are executed in parallel and are unaware of each other. They are separated from the environment input and output actions by the EM. The local input (resp. output) queue of a local execution contains the input (resp. output) items that can be freely consumed (resp. generated) by this local execution. MAP and REDUCE are responsible for respectively the global input queue containing the input items from the external environment, and the global output queue containing the output items filtered by the EM to the environment. When a local execution needs an input item that is not yet ready in its local input queue it will request the help of MAP by emitting an *interrupt signal.* When a local execution generates an output item it emits a signal to request the help of REDUCE.

MAP and REDUCE can autonomously send and, respectively, collect items from local queues. The actions of MAP (resp. REDUCE) on an input (resp. output) request from a local execution depend on the configuration information in the table T_M (resp. T_R). This configuration is based on two privileges: *ask* (a) and *tell* (t). These components of the EM are customized depending on the desired information flow policy.

All local executions with the *tell* privilege on the input channel c can get the real value from the channel c when MAP broadcasts the input item to local executions, otherwise they will get a default value. If a local execution has the *tell* privilege on the output channel c, REDUCE can tell its value to the environment. Otherwise, REDUCE will just replace it with a default value.

If a local execution has the *ask* privilege on the input channel c, then MAP can fetch the input item from the environment upon receiving a signal from a local execution. A local execution with the *ask* privilege on the output channel c can ask REDUCE to start processing outputs from the local executions.

An execution with only the *ask* but not the *tell* privilege in T_R will activate REDUCE to retrieve output items, but REDUCE will not put the value in the external output (i.e. will not tell it to anyone). The execution will have to wait for somebody else with the *tell* privilege on the channel to produce an output.

3 Information flow policies

In this section we briefly present some policies.

Non-Interference. Let $(\pi, I) \Downarrow O$ denote a terminating execution of π that consumes input sequence I and generates output sequence O. Given a security level l (where l is in $\{L, H\}$), $I|_l$ (resp. $O|_l$) returns the projection of the sequence I (resp. O) containing only items at level l. For NI, for two arbitrary input sequences I and I' that are low-equivalent ($I'|_L = I|_L$), the generated outputs O and O' are also low-equivalent ($O'|_L = O|_L$). NI comes in termination-sensitive (TSNI) or termination-insensitive (TINI) flavors.

Definition 1 (TINI). *A program π is* TINI *iff*

$$\forall I, I' : I'|_L = I|_L \wedge (\pi, I) \Downarrow O \wedge (\pi, I') \Downarrow O' \implies O'|_L = O|_L$$

The formal definition of TSNI can be derived from TINI by moving $(\pi, I') \Downarrow O'$ after the implication.

Non-Deducibility. Sutherland defines ND by using two views: the first view corresponds to secret events, and the second view corresponds to observations of attackers at the low level [15]. There are no flows from from the former to the latter if the two views can always be combined. In this way an attacker cannot know whether a particular high input took place, because it can be always replaced by another valid input and still yield a valid execution.

Termination-insensitive ND (TIND) is defined in Def. 2. TIND requires that for any two inputs I and I^*, such that the program terminates with these inputs, there exists another input I^{**}, which is low-equivalent with I ($I|_L = I^{**}|_L$), high-equivalent to I^* ($I^*|_H = I^{**}|_H$), and if the program terminates with I^{**}, the generated output visible to attackers at the low level (L) is not changed. Termination-sensitive ND (TSND) assumes that attackers can observe terminations of executions and the existence of the default view where we replaced input values with default values. If the default values could not be accepted by an execution then it would be possible to deduce that the high information is actually different from the default value.

Definition 2 ((Input-Output) TIND). *A program π is* TIND *iff*

$$\forall I, I^* : (\pi, I) \Downarrow O \wedge (\pi, I^*) \Downarrow O^* \implies (\exists I^{**} : I|_L = I^{**}|_L \wedge I^*|_H = I^{**}|_H \wedge \\ \wedge ((\pi, I^{**}) \Downarrow O^{**} \implies O|_L = O^{**}|_L))$$

The formal definition of TSND can be derived from TIND by requiring that $(\pi, I^{**}) \Downarrow O^{**}$ holds and the execution where all input values have been replaced by default values is always present and terminates.

Removal of Inputs. RI [9] requires that if a possible trace is perturbed by removing all high input items, then the result can be corrected into a possible trace. In our notation, an input (resp output) is a queue of input (resp. output) vectors (see §5). If all high input items in an input I are replaced by default items ($\vec{df}$) or removed, the input can be modified to an input I' such that the program terminates when executing on I' and the generated output will be equivalent at the low level with the original output. $I|_c$ returns an input I' whose items are in I and from channel c.

Definition 3 (RI). *A program π satisfies* RI *iff*

$$\begin{aligned}\forall I, \forall\ values\ of\ val_{def} : (\pi, I) \Downarrow O \implies \exists I' : & I'|_L = I|_L \wedge \forall c, \parallel I'|_c \parallel \leq \parallel I|_c \parallel \ \wedge \\ & \wedge\ I'|_H = (\vec{df})^* \ \wedge \\ & \wedge\ (\pi, I') \Downarrow O' \wedge O'|_L = O|_L\end{aligned}$$

where $\vec{df}$ contains the default value, and $\parallel Q \parallel$ returns the length of Q.

Deletion of Inputs. DI [9] requires that if we perturb a possible trace $t = \beta.e.\alpha$ (there is no high input event in α) by deleting the high input event e, the result can be corrected into a possible trace t' ($t' = \beta'.\alpha'$). Parts β and β' and α and α' are equivalent on the low input events and the high input events; α and α' are also equivalent on low output events. In our notation, if we have an input $I = I_1.\vec{v}.I_2$, where $\vec{v}$ contains a value from a high channel ($\vec{v}[c] \neq \perp$ and $LVL[c] = H$) and in I_2 there are either no high items or only high items with default values ($I_2|_H = (\vec{df})^*$), then this input can be changed by replacing $\vec{v}$ by a default vector ($\vec{df}$). The obtained input can be sanitized by removing existing default high input items in I_2 or adding other default high input items to I_2. The sanitized queue is consumed completely by the program and the output is still low-equivalent to the original output generated with input I ($O'|_L = O|_L$).

Definition 4 (DI). *A program π satisfies* DI *iff*

$$\begin{aligned}&\forall I, \forall\ values\ of\ val_{def} : I = I_1.\vec{v}.I_2 \wedge LVL[c] = H \wedge I_2|_H = (\vec{df})^* \wedge (\pi, I) \Downarrow O \\ &\implies \exists I' : I' = I_1.I_2' \wedge I'|_L = I|_L \wedge I_2'|_H = (\vec{df})^* \wedge (\pi, I') \Downarrow O' \wedge O'|_L = O|_L\end{aligned}$$

where $\vec{v}[c] \neq \perp$ and $\vec{df}$ contains a default value.

4 Implementing the policies

```
if $a \in T_M[i][c]$ then
    input $x$ from $c$
    map$(x, c, canTell(c))$
    map$(val_{def}, c, \neg canTell(c))$
    wake$(isReady(c))$
else
    if $t \notin T_M[i][c]$ then
        map$(val_{def}, c, identical(i))$
        wake$(identical(i))$
    else
        skip
```

(a) MAP for an input from c from $\pi[i]$

```
$x := val_{def}$
if $a \in T_R[i][c]$ then
    retrieve $x$ from $(i, c)$
if $t \in T_R[i][c]$ then
    output $x$ to $c$
clean$(c, identical(i))$
wake$(identical(i))$
```

(b) REDUCE for an output to c from $\pi[i]$

T_M	$\pi[0]$	$\pi[1]$
$LVL[c] = H$	at	$-$
$LVL[c] = L$	t	at

T_R	$\pi[0]$	$\pi[1]$
$LVL[c] = H$	at	$-$
$LVL[c] = L$	$-$	at

Fig. 3. Implementation of NI

Non-Interference. Implementation of NI is in Fig. 3. The EM of NI on a program π needs only two local executions: the high execution ($\pi[0]$) and the low execution ($\pi[1]$). When the low execution needs a high input item, MAP sends a fake value to it. Thus, the execution of the low is independent from high input items consumed by the EM. In addition, only the low execution can send output items to low output channels. Put differently, high input items do not influence consumed low inputs and generated low outputs.

When MAP is activated on signal c from $\pi[i]$ having the ask privilege on c, MAP performs an input action, sends the real value to all local copies having the tell privilege on c, and sends a fake value to others. When MAP is activated on a signal c from $\pi[i]$ that has no privilege on c, MAP sends a fake value to $\pi[i]$ and wakes it up. Function $canTell(c) \triangleq \lambda x.t \in T_M[x][c]$ indicates whether a local copy $\pi[x]$ has the tell privilege on c. A local copy is ready to be waken up if it has received the required input item, $isReady(c) \triangleq \lambda x.EX[x].\mathsf{stt} = \mathsf{S} \wedge EX[x].\mathsf{prg} = \textbf{input } y \textbf{ from } c; \pi \wedge EX[x].\mathsf{in} = I \wedge dequeue(I, c) = (val, I') \wedge val \neq \bot$, where $dequeue(I, c) = (val, I')$ means there is an item from c in I. Function $identical()$ is defined as $identical(i) \triangleq \lambda x.x = i$.

When REDUCE is activated on a signal c from $\pi[i]$, it checks whether $\pi[i]$ has the ask privilege on c ($a \in T_R[i][c]$). If so, REDUCE gets the output value from the local output queue of $\pi[i]$. Otherwise a fake output value is used. REDUCE only sends an output value to c if $\pi[i]$ has the tell privilege on c ($t \in T_R[i][c]$). After that, the output queue of $\pi[i]$ is cleaned and $\pi[i]$ is waken.

In [10] we give a full proof that SME as identified by [5] is captured by our mechanism. In [5] soundness and precision are proved w.r.t a specific scheduler, our proof works for any scheduler respecting the configuration.

We illustrate the execution of the EM on a sample program presented in Fig. 4. The execution of this program requires confidential information about salary and bonus (at lines 2 and 5). This program does not satisfy NI since the desired salary can be sent to public channels (`evil.com` at line 7).

The execution of local executions of the EM is described in Fig. 5 with the input sequence (`cL1` $= \mathbf{T}$) (`cH1` $= M$)(`cH2` $= m$) which means that the position chosen by the applicant is "CEO", his desired salary is M, and the bonus is m. The high and the low copies execute instructions from line 1 to 7. The value

```
input l1 from cL1 //Get the position selected by the applicant.
input h1 from cH1 //Get the desired salary entered by the applicant.
h2 = 0
if l1 then //If the selected position is CEO,
  input h2 from cH2 //Get the bonus from https://goodCompany/getBonus.
output h1 + h2 to cH3 //Show the income to users.
output h1 + h2 to cL2 //Send the income to http://evil.com/.
```

The script gets the desired position chosen by a prospective applicant from a public channel; and retrieves the desired annual salary from a confidential channel. If the chosen position is CEO, the script fetches also the annual bonus from `goodCompany/getBonus`, a confidential channel. Then, it shows the desired salary and the bonus to the applicant via `cH2`, and sends everything to `evil.com`.

Fig. 4. Running Example Program

```
1 input l1 from cL1 //Use T asked by π[1].
2 input h1 from cH1 //Get M from cH1.
3 h2 = 0;
4 if l1 then
5   input h2 from cH2 //Get m from cH2.
6 output h1 + h2 to cH3 //Send M + m to cH3.
7 ~~output h1 + h2 to cL2~~ //The output is ignored.
```

(a) The high execution $\pi[0]$

```
1 input l1 from cL1 //Get T from cL1.
2 input h1 from cH1 //The default value is used.
3 h2 = 0;
4 if l1 then
5   input h2 from cH2 //The default value is used.
6 ~~output h1 + h2 to cH3~~//The output is ignored.
7 output h1+h2 to cL2 //Send ∗ to cL3.
```

(b) The low execution $\pi[1]$

Fig. 5. Executions of local copies for NI

Input to MAP:

	0	1	2
cL1	**T**	$\perp$	$\perp$
cH1	$\perp$	M	$\perp$
cH2	$\perp$	$\perp$	m

Output by REDUCE:

	0	1	2	3	4
cH3	$\perp$	$\perp$	$\perp$	$M+m$	$\perp$
cL2	$\perp$	$\perp$	$\perp$	$\perp$	$*$

Local Executions:

High execution $\pi[0]$:

Local input:

cL1	**T**	$\perp$	$\perp$
cH1	$\perp$	M	$\perp$
cH2	$\perp$	$\perp$	m

Local output:

cH3	$\perp$	$\perp$	$\perp$	$M+m$	$\perp$
cL2	$\perp$	$\perp$	$\perp$	$\perp$	$\bullet$

Low execution $\pi[1]$:

Local input:

cL1	**T**	$\perp$	$\perp$
cH1	$\perp$	$*$	$\perp$
cH2	$\perp$	$\perp$	$*$

Local output:

cH3	$\perp$	$\perp$	$\perp$	$\bullet$	$\perp$
cL2	$\perp$	$\perp$	$\perp$	$\perp$	$*$

$\bullet$ is an output value that is ignored. $*$ is a default value or is calculated based on default values.

Fig. 6. Input and output queues for NI

generated by the output instruction of the high copy (resp. the low copy) at line 7 (resp. line 6) is ignored. To facilitate the presentation we present the contents of the global and local input and output queues in Fig. 6. The global input queue is consumed completely by the execution of the EM. The values sent to `cH3` and `cL2` are respectively $M + m$ and $*$, where $*$ denotes values calculated based on default values. Each column in the table corresponds to an input/output operation. Input and output tables should be read from left to right; columns describe the input/output to each channel at time $t = 0$, $t = 1$, etc.

Non-Deducibility. The configuration of the mechanism of ND requires three local copies. The low execution ($\pi[2]$) can consume only low input items and generate low output item. The high execution ($\pi[0]$) can consume real values from all channels and can send high output items to the environment. The purpose of the shadow execution ($\pi[1]$) is to make sure that low inputs do not determine high inputs. Indeed the shadow execution is

T_M	$\pi[0]$	$\pi[1]$	$\pi[2]$
$LVL[c] = H$	t	at	$-$
$LVL[c] = L$	t	$-$	at

T_R	$\pi[0]$	$\pi[1]$	$\pi[2]$
$LVL[c] = H$	at	$-$	$-$
$LVL[c] = L$	$-$	$-$	at

Fig. 7. Impl. of ND

the only one that can ask for high inputs but only receives dummy low inputs. We used the word shadow as its output are ignored (only legitimate high output from the high is going to see the light). In other words, the low inputs and the high inputs consumed by the EM are independent from each other.

The programs of MAP and REDUCE are the same as the ones of NI. Privileges of the low execution are the same as those of the low execution of NI. The only difference is that the high execution can be told but cannot ask input values and can output its values to high output channels. The shadow execution is the only one that can ask for high input items. Fig. 7 shows configuration of T_M and T_R. Our EM is slightly stronger as it will generate the correct low output even if the high execution might not terminate.

Removal of Inputs. The configuration of RI is in Fig. 8. The EM of RI is similar to the one of NI except the way of handling signals on high input channels from the low execution ($\pi[1]$). To ensure the existence of I' as in the definition, MAP is allowed to ask high input items for the low execution. To ensure that the behaviors visible to attackers do not change, the low execution receives only default high input items and only it can send outputs to low output channels.

```
if $a \in T_M[i][c]$ then
    input $x$ from $c$
    map($x, c, canTell(c)$)
    map($val_{def}, c, \neg canTell(c)$)
    wake($isReady(c)$)
else
    skip
```

(a) MAP for an input from c from $\pi[i]$

T_M	$\pi[0]$	$\pi[1]$
$LVL[c] = H$	at	a
$LVL[c] = L$	t	at

Fig. 8. Implementation of RI

The configuration table T_M is similar to the one of NI except that the low execution has the ask privilege on high input channels. The MAP program is also similar to the one of RI except the cases of handling signals from the low execution on high input channels. In these cases, MAP performs an input action, sends the read value to the high, and send a default value to the low. Functions $canTell(c)$, $isReady(c)$ and $identical(i)$ are as in the ones in NI.

Deletion of Inputs. DI is enforced with the idea that whenever the high execution ($\pi[0]$) requests a high input item, this execution will be cloned. The clones have to reuse low input items asked by the low execution ($\pi[1]$), will not receive real values from high channels and cannot send output to the environment. As in NI, the low execution can only receive fake high input values.

Implementation of EM of DI is presented in Fig. 9.

```
if $LVL[c] == H$ and $i == 0$ then
    clone($identical(i), PRIV_{T_M}, PRIV_{T_R}$)
if $a \in T_M[i][c]$ then
    if $t \in T_M[i][c]$ then
        input $x$ from $c$
        map($x, c, canTell(c)$)
        map($val_{def}, c, \neg canTell(c)$)
        wake($isReady(c)$)
    else
        map($val_{def}, c, identical(i)$)
        wake($identical(i)$)
else
    skip
```

(a) MAP for DI for an input from c from $\pi[i]$

T_M	$\pi[0]$	$\pi[1]$	$\pi[i] > 1$
$LVL[c] = H$	at	$-$	$-$
$LVL[c] = L$	t	at	t

T_R	$\pi[0]$	$\pi[1]$	$\pi[i] > 1$
$LVL[c] = H$	at	$-$	$-$
$LVL[c] = L$	$-$	at	$-$

Fig. 9. Implementation of DI

$$\text{INP}\ \frac{\pi = \textbf{input}\ x\ \textbf{from}\ c \qquad I = \vec{v}.I' \qquad \vec{v}[c] \neq \bot}{\begin{array}{c}\mathit{\Delta}, \mathsf{prg}{:}\pi, \mathsf{mem}{:}m, \mathsf{in}{:}I \\ \twoheadrightarrow \mathit{\Delta}, \mathsf{prg}{:}\mathbf{skip}, \mathsf{mem}{:}m[x \mapsto \vec{v}[c]], \mathsf{in}{:}I'\end{array}} \qquad \text{OUTP}\ \frac{\pi = \textbf{output}\ e\ \textbf{to}\ c \qquad \vec{v} = \vec{\bot}[c \mapsto m(e)]}{\begin{array}{c}\mathit{\Delta}, \mathsf{prg}{:}\pi, \mathsf{out}{:}O \\ \twoheadrightarrow \mathit{\Delta}, \mathsf{prg}{:}\mathbf{skip}, \mathsf{out}{:}O.\vec{v}\end{array}}$$

Fig. 10. Semantics of input and output instructions of programs

The program of REDUCE is identical to the one in NI. The EM of DI requires more than two local executions. Only the high execution $\pi[0]$ can ask for and get the high input items, other local executions will use default values. Each time the high execution asks a high input item, it is cloned. In Fig. 9 the configuration of the clones for input and output is presented in respectively T_M and T_R in the columns with title $\pi[i] > 1$; These columns are the privilege templates for $PRIV_{T_M}$ and $PRIV_{T_R}$ in clone instruction in Fig. 9a. As in NI, only the low execution $\pi[1]$ can ask for low input items and generate low output items; other local executions will reuse the low input items retrieved by the low execution. Functions $canTell(c)$, $isReady(c)$ and $identical(i)$ are as in the ones in NI.

5 Semantics

Semantics of controlled programs. Our model language is close to the one used in the SME paper [5]. Valid values in this language are boolean values (**T** and **F**) or non-negative integers. A program π is an instruction described in Fig. 2a where π, e, x, and c are meta-variables for respectively instructions, expressions, variables, and input/output channels. Since a program is just a sequence of instructions (i.e. a complex instruction itself), we will use program and instruction interchangeably when referring to complex instructions. We model an input (output) item as a vector $\vec{v}$ and define input (output) of program instances as queues I, O so that $\vec{v}.I$ (resp. $\vec{v}.O$) adds the element $\vec{v}$ to the queue. We use vectors of channel to accommodate forms in which multiple fields are submitted simultaneously but are classified differently (e.g. credit card numbers vs. user names). Given a vector $\vec{v}$ and a channel c, the *value of the channel* is denoted by $\vec{v}[c]$. To simplify the formal presentation, in the sequel w.l.o.g. we assume that each input and output operation only affect one channel at a time. Thus, for each vector, there is only one channel c such that $\vec{v}[c] \neq \bot$.

To define an execution configuration, we use a set of labelled pairs. A labelled pair is composed by a label and an object and in the form label:*object*. The label is attached to the *object* in order to differentiate this object from others, so each label occurs only once. An *(execution) configuration* of a program is a set $\{\mathsf{prg}{:}\pi, \mathsf{mem}{:}m, \mathsf{in}{:}I, \mathsf{out}{:}O\}$, where π is the program to be executed, m is the memory (a function mapping variables to values), I (resp. O) is the queue of input (resp. output) vectors. The operational semantics of the input and output instructions of the model language is the natural one. Fig. 10 illustrates some examples. See also [5] for similar one and [10] for detail. The conclusion part of each semantic rule is written as $\mathit{\Delta}, \mathit{\Gamma} \Rightarrow \mathit{\Delta}, \mathit{\Gamma}'$, where $\mathit{\Delta}$ denotes the elements of the execution configuration that are unchanged upon the transition. We abuse the notation $m(.)$ and use it to evaluate expressions to values. When an output command sends a value to the channel c, an output vector $\vec{v} = \vec{\bot}[c \mapsto val]$ is inserted into the output queue, where $\vec{v}$ is the vector with all undefined channels, except c that is mapped to $m(e)$, so $\vec{v}[c'] = \bot$ for all $c' \neq c$ and $\vec{v}[c] = m(e)$.

Semantics of the Enforcement Mechanism. A *configuration of an EM* is a set $\{\mathsf{t_m}{:}T_M, \mathsf{t_r}{:}T_R, \mathsf{top}{:}TOP, \mathsf{map.prg}{:}\pi_M, \mathsf{map.mem}{:}m_M, \mathsf{red.prg}{:}\pi_R, \mathsf{red.mem}{:}m_R, \mathsf{in}{:} I, \mathsf{out}{:}O, \bigcup_i \mathsf{LECS}_i\}$, where T_M and T_R are configuration tables for respectively MAP and REDUCE, TOP is the index of the top of the stack of configurations of local executions EX, π_M and m_M (resp. π_R and m_R) are the program to be executed and the memory of MAP (resp. REDUCE), I and O are respectively the input and output queues of the EM, and LECS_i is the configuration of the i-th local execution.

For the initial configuration, all local input and output queues will be empty, all local executions will be in the executing state, and skip is the only instruction in MAP and REDUCE programs. The EM terminates when all local executions, MAP and REDUCE programs terminate, and the global input queue is consumed completely.

The semantics of EM is the interleaving of concurrent atomic instructions of the various programs so each transition rule either by a local execution, by MAP, or by REDUCE is a step of the EM as a whole.

Local Executions. Each local execution is identified by a unique identifier i, which is its number on stack EX. A local copy can be in one of two states: **E** (Executing) or **S** (Sleeping). A local copy moves from **E** to **S** when it needs an input item that is not available in its local queue or when it generates an output item. A local copy moves from **S** to **E** when the required input item is ready or its output item is consumed.

A configuration of i-th local copy is $\mathsf{LECS}_i \triangleq \{EX[i].\mathsf{stt} : st, EX[i].\mathsf{int} : s, EX[i].\mathsf{prg} : \pi, EX[i].\mathsf{mem} : m, EX[i].\mathsf{in} : I, EX[i].\mathsf{out} : O\}$, where st is its state, s is a signal, π, m, I, and O are as in configuration of controlled programs, EX is the global stack of local execution. The initial configuration of i-th local copy is $\{EX[i].\mathsf{stt}{:}\mathbf{E}, EX[i].\mathsf{int}{:}\bot, EX[i].\mathsf{prg}{:}\pi, EX[i].\mathsf{mem}{:}m_0, EX[i].\mathsf{in}{:}\epsilon, EX[i].\mathsf{out}{:}\epsilon\}$. A local copy terminates if there is only a skip instruction to be executed.

The semantics of assignment, composition, if, while, skip instructions is essentially identical to the one of the controlled programs. The only difference is the explicit condition that the local state must be **E**. When the input instruction of $\pi[i]$ is executed and the required input item is not in the local input queue ($dequeue(I, c) = (\bot, I')$), $\pi[i]$ emits a signal c and moves to a sleep state (rule LINP2 in Fig. 11). Otherwise, the first available item will be consumed. A signal c is generated when the output instruction is executed (rule LOUTP in Fig. 11).

MAP. In addition to the instructions in Fig. 2a (except the output instruction), the program π_M is also composed by instructions in Fig. 2b, where $PRED[\,] \triangleq$

$$\text{LINP2}\ \frac{EX[i].\mathsf{stt} = \mathbf{E} \qquad EX[i].\mathsf{prg}{:}\pi = \mathbf{input}\ x\ \mathbf{from}\ c \qquad dequeue(I, c) = (\bot, I')}{\varDelta, EX[i].\mathsf{stt}{:}\mathbf{E}, EX[i].\mathsf{int}{:}\bot \Rightarrow \varDelta, EX[i].\mathsf{stt}{:}\mathbf{S}, EX[i].\mathsf{int}{:}c}$$

$$\text{LOUTP}\ \frac{EX[i].\mathsf{stt} = \mathbf{E} \qquad \pi = \mathbf{output}\ e\ \mathbf{to}\ c \qquad EX[i].\mathsf{mem} = m \qquad \vec{v} = \vec{\bot}[c \mapsto m(e)]}{\begin{array}{c}\varDelta, EX[i].\mathsf{stt}{:}\mathbf{E}, EX[i].\mathsf{int}{:}\bot, EX[i].\mathsf{prg}{:}\pi, EX[i].\mathsf{out}{:}O \\ \Rightarrow \varDelta, EX[i].\mathsf{stt}{:}\mathbf{S}, EX[i].\mathsf{int}{:}c, EX[i].\mathsf{prg}{:}\mathbf{skip}, EX[i].\mathsf{out}{:}O.\vec{v}\end{array}}$$

Fig. 11. Semantics of input and output instructions of controlled $\pi[i]$

$$\text{MAP}\ \frac{\begin{array}{c}\pi_M = \mathbf{map}(e, c, PRED[\ ]) \qquad m = \mathsf{map.mem} \qquad S = \{i \in \{0, \dots, TOP\} : PRED[i]\} \\ \mathsf{LECS} = \bigcup_{i \in S} \{EX[i].\mathsf{in}{:}I\} \qquad \vec{v} = \vec{\bot}[c \mapsto m(e)] \qquad \mathsf{LECS}' = \bigcup_{i \in S} \{EX[i].\mathsf{in}{:}I.\vec{v}\}\end{array}}{\varDelta, \mathsf{map.prg}{:}\pi_M, \mathsf{LECS} \Rightarrow \varDelta, \mathsf{map.prg}{:}\mathbf{skip}, \mathsf{LECS}'}$$

$$\text{RETR}\ \frac{\pi_R = \mathbf{retrieve}\ x\ \mathbf{from}\ (i, c) \qquad EX[i].\mathsf{out} = O \qquad dequeue(O, c) = (val, O') \qquad val \neq \bot}{\varDelta, \mathsf{red.prg}{:}\pi_R, \mathsf{red.mem}{:}m \Rightarrow \varDelta, \mathsf{red.prg}{:}\mathbf{skip}, \mathsf{red.mem}{:}m[x \mapsto val]}$$

Fig. 12. Semantics of map and retrieve instructions of MAP and REDUCE

$\lambda x.Pred(x)$ is a meta-variable for predicates. The evaluation of the predicate $PRED[\ ]$ on $\pi[i]$ is denoted as $PRED[i]$.

The execution of map, wake, or clone instruction is applied simultaneously to all local executions $\pi[i]$ such that $PRED[i]$ is true. For map, the value of expression e (which is considered from c) is sent to the input queues of all $\pi[i]$. The semantics of map instruction is described in Fig. 12. For wake, all local executions $\pi[i]$ are awaken and interrupt signals in their configurations are removed. For clone, the configuration of each $\pi[i]$ is cloned. The list $PRIV_{T_M}$ (resp. $PRIV_{T_R}$) is an input (resp. output) privilege template for clones which varies depending on the enforced property. We give an example of such templates in §4, where the enforced property requires cloning.

The initial configuration of MAP is $\{\mathsf{map.prg}{:}\pi_M, \mathsf{map.mem}{:}m_0\}$. The execution of MAP terminates if skip is the only instruction in the MAP program. MAP is activated when the previous execution of MAP has terminated, and there is a local execution asking for help for an input item.

REDUCE. Except the input instruction, in addition to the instructions in Fig. 2a, the program of REDUCE may contain instructions in Fig. 2c. The execution of retrieve instruction reads the value from the output queue of $\pi[i]$ and stores it into x. The execution of clean instruction is applied to all $\pi[i]$ such that $PRED[i]$ is true. This instruction removes the first output item to c from O of $\pi[i]$. The execution of the wake instruction is similar to the one of MAP. Configuration, activation and termination of REDUCE are similar to the ones of MAP. The semantics of retrieve instruction is shown in Fig. 12 where $dequeue(O, c)$ returns a first item to c in O.

6 Formal Properties

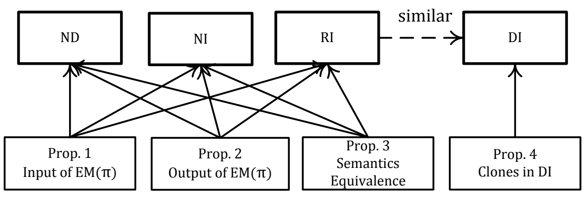


Fig. 13. Proof Strategy for Soundness

[The full versions of the proofs are available in [10].] The soundness property states that the EM correctly enforces the desired policy on an arbitrary program. Our notion of soundness is taken from [5, 4] and is close to the one used in [8]. It has some known limitations (see [11] for a different definition) but we retained it because it is widely used and understood. Soundness does not hold for EMs of termination-sensitive properties because one local copy might terminate but the others might not. Thus, the whole EM does not terminate.

Theorem 1 (Soundness of Enforcement). *For all programs π, each EM executed on π in Tab. 1 satisfies the corresponding policy, except for termination-sensitive policies.*

The proof strategy of soundness is sketched in Fig. 13. Prop. 1 states that the input handling in MAP is correct w.r.t. the specification: e.g., we prove that for NI, MAP only asks input items from the environment for high input requests from the high execution. Prop. 2 states that the output handling by REDUCE is correct w.r.t. to the specification: e.g., only the high execution sends items to high output channels. Prop. 3 states that the semantics of controlled programs and the semantics of local executions are equivalent (for I_1 and I_2, which coincide for all channels, the execution of the original program on I_1 and the execution of a local copy on I_2 yield the same output queues).

To prove the soundness theorem for NI and ND we perform case-based reasoning showing that outputs produced by EMs satisfies the respective definitions. This proof strategy is also used to prove the soundness theorem for RI. For DI, we need another proposition (Prop. 4) stating that the clones do not influence the consumed inputs and the generated outputs of the EM.

The notion of precision for enforcement of a property is taken from [5, 4]. The intuition is that the EM does not change the visible behavior of a program that is secure with respect to the property (and in particular the I/O behaviour on specific channels).

Definition 5. *An EM is* precise *w.r.t a property, if for any program π satisfying the property, and for every input I, where $(\pi, I) \Downarrow O$, the actually consumed input I^* and the actual output O^* of the EM, regardless of the order of executing local copies, are s.t. EM terminates and $I^*|_c = I|_c$ and $O^*|_c = O|_c$ for all channels c.*

Theorem 2 (Precision of Enforcement). *Each EM in Tab. 1 is precise w.r.t. the corresponding policy except for termination-insensitive policies.*

Fig. 14 shows the proof strategy for precision. We prove simple properties regarding the correct handling of interrupt signals (Prop. 5 and Prop. 6). We show that from the input of the high execution we can reconstruct the original global input (Prop. 7). The proof of the precision theorem of the EM of NI (resp. ND) follows directly from Lem. 1 (resp. Lem. 2). Lem. 1 shows that if a program π satisfies TSNI, terminates, and all local executions consume input correctly, then the consumed input of the mechanism is I^* where $I|_c = I^*|_c$ for all c. The proofs of the precision theorem of EMs of RI and DI are similar.

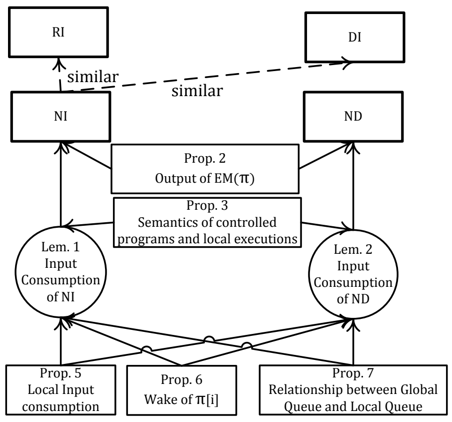

RI
DI
similar
similar
NI
ND
Prop. 2 Output of EM(π)
Prop. 3 Semantics of controlled programs and local executions
Lem. 1 Input Consumption of NI
Lem. 2 Input Consumption of ND
Prop. 5 Local Input consumption
Prop. 6 Wake of π[i]
Prop. 7 Relationship between Global Queue and Local Queue

Fig. 14. Proof Strategy for Precision

Precision does not hold for mechanisms of termination-insensitive properties. For a program satisfying a termination-insensitive property, its execution on an input might terminate, while execution on the other inputs as in the definition of the property might not. Thus, there is a case that the high copy might terminate but other executions might not.

7 Related Works and Conclusions

Information flow policies can be enforced by many approaches [13, 12, 3]. Our choice of using the multi-execution approach, despite its performance overhead, was dictated by its advantages over the static and dynamic information flow analysis techniques. Furthermore, the multi-execution approach is also practical as demonstrated in [4], where SME, an instance of this approach, is implemented in FireFox. The implementation introduces a noticeable performance overhead but not prohibitive and the implementation works with most existing web sites.

SME [5] has inspired many researchers to push further investigation of this technique. The influence of the order of executing local copies on timing and termination channels is investigated in [8]. Stronger notions of precision are investigated in [11, 16]. Our current proposal does not address timing and termination channels, and does not offer the same precision guarantees. However, our proposal can be further extended by using the techniques proposed in [8, 11, 16]. The focus of our paper is to develop a programmable framework that is capable of handling different information flow properties.

SME-based EMs of declassification policies are proposed in [1, 11]. Our framework can be instantiated to enforce stateless declassification policies like the one in [11] where the existence of the high input items can be released. The configuration of this policy is similar to the one of RI except that the low does not have the ask privilege on high input channels. To enforce stateful declassification policies in which the physical locations of release are specified [14], one possible approach is to introduce declassify operators as in [1, 11]. However, by doing this we lose one advantage of SME which treats controlled programs as black boxes.

We presented a programmable framework that can enforce multiple information flow properties via running several copies of a program. The framework is instantiated for enforcing non-interference (NI) [5], non-deducibility (ND) [15], removal of inputs (RI) and deletion of inputs (DI) [9]. For these properties we formally proved soundness and precision of enforcement.

The framework uses the MAP and REDUCE components to interact with the environment: all input and output actions are mediated by these two components. Local executions consume different inputs (real input values or default ones) fed by MAP, depending on their privileges in the table T_M; for each channel the outputs are fetched by REDUCE from the dedicated execution (which has the corresponding privilege in the table T_R).

Acknowledgements This work is partly supported by the project EU-IST-NOE-NESSOS.

References

1. T. H. Austin and C. Flanagan. Multiple facets for dynamic information flow. *SIGPLAN Not.*, 47(1):165–178, Jan. 2012.
2. M. Balliu, M. Dam, and G. L. Guernic. Encover: Symbolic exploration for information flow security. In *Proc. of CSF 2012*, pages 30–44, 2012.
3. G. Barthe, J. M. Crespo, D. Devriese, F. Piessens, and E. Rivas. Secure multi-execution through static program transformation. In *Proc. of FMOODS/FORTE 2012*, 2012.
4. W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens. Flowfox: a web browser with flexible and precise information flow control. In *Proc. of CCS 2012*, 2012.
5. D. Devriese and F. Piessens. Noninterference through secure multi-execution. In *Proc. of IEEE S&P 2010*, 2010.
6. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *J. of Comp. Sec.*, 3:5–33, 1994.
7. J. Goguen and J. Meseguer. Security policies and security models. In *Proc. of IEEE S&P'82*, 1982.
8. V. Kashyap, B. Wiedermann, and B. Hardekopf. Timing- and termination-sensitive secure information flow: Exploring a new approach. In *Proc. of IEEE S&P*, 2011.
9. H. Mantel. Possibilistic definitions of security - an assembly kit. In *Proc. of CSFW 2000*, 2000.
10. M. Ngo, F. Massacci, and O. Gadyatskaya. MAP-REDUCE runtime enforcement of information flow policies. Availabe as ArXiv report `http://arxiv.org/abs/1305.2136`. Technical Report DISI-13-019, University of Trento, 2013.
11. W. Rafnsson and A. Sabelfeld. Secure multi-execution: fine-grained, declassification-aware, and transparent. In *Proc. of CSF 2013*, 2013.
12. A. Russo and A. Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *Proc. of CSF 2010*, 2010.
13. A. Sabelfeld and A. Myers. Language-based information-flow security. *J. on Selected Areas in Comm.*, 21(1):5 – 19, 2003.

14. A. Sabelfeld and D. Sands. Declassification: Dimensions and principles. *J. on Comput. Secur.*, 17(5):517–548, Oct. 2009.
15. D. Sutherland. A model of information. In *Proc. of NCSC'86*, 1986.
16. D. Zanarini, M. Jaskelioff, and A. Russo. Enforcement of confidentiality for reactive systems. In *Proc. of CSF 2013*, 2013.

On the Stackelberg fuel pricing problem*

Cosimo Vinci[1] and Vittorio Bilò[2]

[1] Gran Sasso Science Institute, L'Aquila - Italy
cosimo.vinci.gssi@gmail.it

[2] Department of Mathematics and Physics "Ennio De Giorgi", University of Salento, Provinciale Lecce-Arnesano, P.O. Box 193, 73100 Lecce - Italy
vittorio.bilo@unisalento.it

Abstract. We consider the Stackelberg fuel pricing problem in which a company has to decide the fuel selling price at each of its gas stations in order to maximize its revenue, assuming that the selling prices of the competitors and the customers' preferences are known in advance. We show that, even in the basic case in which the road network is modeled by an undirected planar graph and the competitors discriminate on two different selling prices only, the problem is APX-hard. On the positive side, we design a polynomial time algorithm for instances in which the number of gas stations owned by the company is constant, while, in the general case, we show that the single-price algorithm (which provides the best-known solutions for essentially all the Stackelberg pricing problems studied in the literature up to date) achieves an approximation ratio which is logarithmic in some parameters of the input instance. This result, in particular, is tight and holds for a much more general class of Stackelberg network pricing problems.

1 Introduction

A fundamental decisional process in many business activities concerns how to set the selling prices so as to maximize its own revenue, once knowing the selling prices of the competitors and the customers' preferences. The scenario in which the latter are implicitly defined in terms of some optimization problem are usually referred to as *Stackelberg pricing problems* [15]. These problems can be modeled as multi-player one-round games in which there is a special player, the *leader*, while all the others are *followers*. The first action is undertaken by the leader who decides on some parameters (e.g., the selling prices) and then the followers respond by deciding their actions. Each follower adopts, as her action, the optimal solution of a certain optimization problem (e.g., satisfying her own demand at the minimum cost) which depends on some of the parameters fixed by the leader and on some other values on which the leader has no control (e.g., the selling prices of the competitors). Since the followers' choices influence, in

* This work was partially supported by the PRIN 2010–2011 research project ARS TechnoMedia: "Algorithmics for Social Technological Networks" funded by the Italian Ministry of University.

turn, the leader's revenue, the determination of the best possible action for the leader often results in a challenging algorithmic problem.

A considerable research attention, see [2–8, 11, 12, 14], has been devoted in the last years to the study of Stackelberg network pricing problems based on some fundamental (polynomial time solvable) optimization problems, such as shortest paths, shortest path trees and minimum spanning trees. In this paper, we study the *Stackelberg fuel pricing problem* (SFPP) which is a Stackelberg network pricing problem based on the *gas station problem* (GSP): an optimization problem introduced in [9] to model situations in which drivers have to go from one location to another and have to decide where to fill their cars with fuel so as to minimize the travel cost, once knowing the fuel prices at the various gas stations along the road network.

Model and Notation. For a positive integer k, let $[k]$ denote the set $\{1, \dots, k\}$. A *road network* is an edge-weighted directed graph $G = (V, E, w)$, with $|V| = n$, $|E| = m$ and $w : E \to \mathbb{R}_{\geq 0}$ such that, for each $e = (u, v) \in E$, $w(e)$ specifies the amount of fuel (expressed in gallons) needed to go from u to v.

An instance (G, s, t, S, p) of the GSP is defined by a road network G, a pair of nodes $s, t \in V$, a set of nodes $S \subseteq V$ and a function $p : S \to \mathbb{R}_{\geq 0}$. The set of nodes S represents the locations of gas stations and the function p models the selling prices (per gallon) at each of the gas stations. There is a driver who needs to go from s to t. For the sake of simplicity, we assume that the driver's car is equipped with a tank of unlimited capacity and that $s \in S$, so that the driver can fill with as much fuel as she wants at price $p(s)$ when starting her trip. The driver wants to determine the best possible *itinerary*, that is, which (s, t)-path to drive through and which gas stations to stop at for fueling so as to minimize the total fuel cost[3].

An instance $(G, (s_i, t_i, \lambda_i)_{i \in [k]}, L, C, p_C, e)$ of the SFPP is defined by a road network G, k source-destination pairs (s_i, t_i) with an associated integer weight $\lambda_i \geq 1$ for each $i \in [k]$, two sets $L, C \subseteq V$, a function $p_C : C \to \mathbb{R}_{\geq 0}$ and a value $e \geq 0$. The sets of nodes L and C represent the locations of gas stations: the gas stations located at nodes in L are owned by the leader, while those located at nodes in C are owned by her competitors, so that the function p_C models the selling prices established by the competitors at each of their gas stations[4]. Note that we do not require L and C to be disjoint, i.e., either the leader and one of her competitors may own a gas station at the same location. The leader buys (or produces) the fuel at price e per gallon. There are k types of drivers (the followers) such that, for each $i \in [k]$, the driver of type i wants to go from s_i to t_i along the cheapest path in G, where λ_i denotes the number of drivers of type i, that is, how many drivers want to go from s_i to t_i. Once the leader has established a pricing function $p_L : L \to \mathbb{R}_{\geq 0}$ defining the fuel prices at her own gas stations, each follower of type $i \in [k]$ determines her action by

[3] The amount of fuel bought at each gas station s is implicitly defined by the minimum distance between s and the successive station chosen for fueling

[4] In the case in which more than one competitor owns a gas station in a given location v, $p_C(v)$ will denote the cheapest fuel price among them.

solving the instance $(G, s_i, t_i, L \cup C, p)$ of the GSP, in which, for each $v \in L \cup C$, $p(v) := \min\{p_L(v), p_C(v)\}$. The leader has to determine the pricing function $p_L : L \to \mathbb{R}_{\geq 0}$ providing her with the highest possible revenue. To this aim, we make the following simplifying assumption which is common in the setting of Stackelberg pricing problems: when the gas station problem has more than one optimal solution, the follower always chooses the one providing the leader with the highest revenue[5]. Furthermore, we assume that $s_i \in C$ $\forall i \in [k]$, otherwise, either the problem is not feasible or the revenue is unbounded.

More formally, given a node $v \in V$ and a pricing function p_L for the leader, let $q(p_L, v) := 1_{p_L(v) \leq p_C(v)}$. For a pair of nodes $u, v \in V$, let $d(u, v)$ be the distance from u to v in G and $\mathcal{B}(u, v)$ be the set of itineraries connecting u to v, that is, the set of sequences of nodes $[u = v_{i_1}, v_{i_2}, \ldots, v_{i_r} = v]$ such that $v_{i_s} \in L \cup C$ for each $s \in [r-1]$. Set $p_s := p(v_{i_s})$, $d_s := d(v_{i_s}, v_{i_{s+1}})$ and $q_s := q(p_L, v_{i_s})$ for each $s \in [r-1]$. Given p_L and $B \in \mathcal{B}(u, v)$, a driver choosing itinerary B experiences a cost $c(p_L, B)$ and yields a contribution $g(p_L, B)$ to the leader's revenue which are defined as follows:

$$c(p_L, B) := \sum_{s=1}^{r-1} p_s \cdot d_s, \quad g(p_L, B) := \sum_{s=1}^{r-1} (p_s - e) \cdot d_s \cdot q_s.$$

Let $cg(p_L, B) = (c(p_L, B), g(p_L, B))$ and define a total ordering relation $\prec$ on $\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ such that $[x_1, y_1] \prec [x_2, y_2] \Leftrightarrow x_1 < x_2 \vee \{x_1 = x_2 \wedge y_1 > y_2\}$. Let $\mathcal{B}^*(u, v) \subseteq \mathcal{B}(u, v)$ be the set of itineraries $B \subseteq \mathcal{B}(u, v)$ minimizing $cg(p_L, B)$ according to the ordering relation $\prec$. Let $cg(p_L, u, v) := [c(p_L, u, v), g(p_L, u, v)] := \min_{B \in \mathcal{B}(u,v)} cg(p_L, B)$. Given a driver of type i, we have that $c(p_L, s_i, t_i)$ is the cost of her cheapest path, and $g(p_L, s_i, t_i)$ is her contribution to the leader's revenue, so that $g(p_L) := \sum_{i=1}^{k} \lambda_i \cdot g(p_L, s_i, t_i)$ is the leader's total revenue that has to be maximized. Let $p_M := \max_{i \in [k]} p_C(s_i)$. It is easy to prove that, if we set $p_L(v) > \min\{p_M, p_C(v)\}$ for some $v \in L$, no driver will stop at station v for fueling. Similarly, whenever $p_C(v) > p_M$ for some $v \in C$, no driver will stop at station v for fueling. Therefore, we can suppose without loss of generality that $p_C(v) \leq p_M$ $\forall v \in C$, the leader's revenue $g(p_L)$ is bounded and, in order to be maximized, p_L can be chosen in such a way that $e \leq p_L(v) \leq p_C(v)$ $\forall v \in L$.

Related Work. The GSP has been introduced by Khuller, Malekian and Mestre in [9]. They give polynomial time algorithms for the basic problem and for some of its variations.

Briest, Hoefer and Krysta author an influential work [6] on Stackelberg network pricing problems which widely generalizes previous results in the field. In particular, they consider the case in which the edges of the network are partitioned into two sets: the set of *fixed-price edges* and that of *priceable edges*, with the latter owned by the leader. Each follower buys a subnetwork of minimum cost and so the leader wants to assign suitable prices to the priceable edges so

[5] In fact, if this is not the case, the leader can decrease some selling price of a negligible amount so as to achieve almost the same revenue as in the case in which the assumption holds.

as to maximize her revenue. They study the approximation guarantee of the *single-price algorithm* in this general class of problems. This algorithm, which assigns the same (suitably computed) price to all priceable edges, has been first analyzed in [7] for the case of a single follower buying a minimum spanning tree. Briest, Hoefer and Krysta show that, for the case of a single follower, the approximation guarantee is $(1+\epsilon)H_h$, where $\epsilon > 0$ is an arbitrary value, h is the number of priceable edges and H_i is the ith harmonic number, while, for the case of k followers, it becomes $(1+\epsilon)(H_h + H_k)$. Finally, when the followers may have different weights, they show that the single-price algorithm achieves an approximation guarantee of $(1+\epsilon)h^2$ and also provide a lower bound of $O(h^{\epsilon})$ on the approximability of the problem.

Determining whether there are approximation algorithms better than the single-price one is, perhaps, the most important open problem in this field of research. In fact, while the performance of this algorithm remains essentially the same even when instantiated to specific optimization problems such as shortest paths, shortest path trees and minimum spanning trees, the impossibility results known so far in these cases only refer to APX-hardness, see [3, 5, 7].

Our Contribution. We show that the SFPP is APX-hard even in the basic case in which the road network is modeled by an undirected planar graph and the competitors discriminate on two different selling prices only, by means of a reduction from the maximum independent set problem on cubic graphs. This reduction, however, requires that $|L|$ is a non-constant value. This assumption is essential, anyway, since, for the case in which $|L| = O(1)$, we show that the problem can be solved in polynomial time.

We stress that the SFPP does not fall within the scope of the Stackelberg network pricing problems defined by Briest, Hoefer and Krysta in [6] and that the presence of additional parameters in the definition of the problem (in particular, the edge-weights) makes the performance of the single-price algorithm unlikely to be uninfluenced by the characteristics of the road network given in input. To this aim, we define a general class of Stackelberg network pricing problems which extends the one given by Briest, Hoefer and Krysta and includes the SFPP. For this class of problems, we show that the single-price algorithm provides an approximation guarantee which is logarithmic in some parameters of the input instance (see the claim of Theorem 4 for the exact characterization) and that this bound is tight.

Due to space limitations, some details and proofs have been removed.

2 Complexity Results

We first show that the SFPP is APX-hard even under some restrictions.

Theorem 1 *The* SFPP *is APX-hard even when p_C assigns only two different prices and G is an undirected planar graph as long as $|L| \in \Theta(n)$.*

Proof. We consider a polynomial reduction from the Maximum Indipendent Set Problem on cubic graphs (MISP). Given a cubic graph (U, F), with $|U| = n$

and $|F| = m$, and an arbitrary value $\theta \in]0, 1/2]$ polynomially representable with respect to n, consider an instance $(G, (s_i, t_i, \lambda_i)_{i\in[k]}, L, C, p_C, e)$ of the SFFP that we define incrementally as follows. Suppose that the sets V, E, L and C are initially empty. Fix an arbitrary orientation on the edges in U, insert a node v^* in V with $p_C(v^*) = 1 + 3\theta$, then, $\forall i \in [n]$, insert in V three nodes $\overline{v}_i^1$, $\overline{v}_i^2$ and $\overline{v}_i^3$ with $p_C(\overline{v}_i^1) = p_C(\overline{v}_i^3) = 1 + 3\theta$ and $p_C(\overline{v}_i^2) = 4\theta$; $\overline{v}_i^1$ belongs to L. Then, $\forall i \in [n]$, insert in E three edges $\overline{e}_i^1 = \{\overline{v}_i^1, \overline{v}_i^2\}$, $\overline{e}_i^2 = \{\overline{v}_i^2, v^*\}$ and $\overline{e}_i^3 = \{\overline{v}_i^3, \overline{v}_i^1\}$ with $\omega(\overline{e}_i^1) = \theta$, $\omega(\overline{e}_i^2) = 1/2 - \theta$ and $\omega(\overline{e}_i^3) = 1 - \theta$. $\forall e_j = (u_i, u_h) \in F$, denote $\overline{v}_i^1$ with v_j^1, $\overline{v}_i^2$ with v_j^2, $\overline{v}_h^2$ with v_j^3, $\overline{v}_h^1$ with v_j^4 and $\overline{v}_h^3$ with v_j^5. There is a driver $(\overline{v}_i^1, \overline{v}_i^2)$ $\forall i \in [n]$, and a driver (v_j^1, v_j^5) $\forall j \in [m]$. Finally, set $e = 3\theta$.

We stress that the constructed road network G is a planar graph.

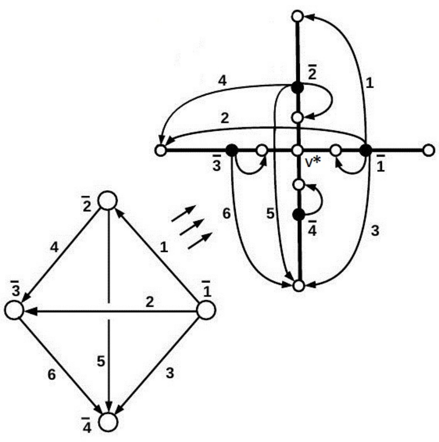


Fig. 1. We consider, for example, the polynomial reduction applied to a complete graph (U, F) with $|U| = 4$, depicted on the left. In (U, F) the edges are arbitrary oriented and there is an enumeration of vertices and edges such that the vertex numbers are overlined, whereas the edge numbers are not. On the right, we have the road network G obtained from U. The index $\overline{i}$ indicates the driver $(\overline{v}_i^1, \overline{v}_i^2)$ $\forall i \in [4]$, and the index j indicates the driver (v_j^1, v_j^5) $\forall j \in [6]$. The starting and the arriving nodes of the source-destination pair associated with every driver are depicted trough an arrow.

The following lemmas hold.

Lemma 1 *Consider the instance obtained by the reduction from (U, F). Given a pricing function p_L, let p'_L be the pricing function such that $p'_L(\overline{v}_i^1) = 4\theta$ if $p_L(\overline{v}_i^1) \leq 4\theta$ and $p'_L(\overline{v}_i^1) = 1 + 3\theta$ otherwise. It holds that $g(p'_L) \geq g(p_L)$.*

Proof (Sketch). We first prove that, independently from p_L, the cheapest path for driver $(\overline{v}_i^1, \overline{v}_i^2)$ is $[\overline{v}_i^1, \overline{v}_i^2]$ $\forall i \in [n]$ and that the cheapest path for driver (v_j^1, v_j^5) is $[v_j^1, v_j^2, v^*, v_j^3, v_j^4, v_j^5]$ $\forall j \in [m]$. At this point, we prove that, by setting to 4θ all the prices not bigger than 4θ and to $1+3\theta$ the remaining ones (thus obtaining p'_L), the leader gets a revenue of $g(p'_L) \geq g(p_L)$. □

Lemma 2 *Given the pricing function p'_L of the previous lemma, there exists a pricing function p^*_L which can be obtained in polynomial time from p'_L such that $p^*_L(\overline{v}_i^1) = 4\theta$ if exists $f_j \in \delta^+_{(U,F)}(u_i)$ such that $p'_L(v_j^1) = p'_L(v_j^4) = 1 + 3\theta$, and $p^*_L(\overline{v}_i^1) = p'_L(\overline{v}_i^1)$ otherwise. Moreover, it holds that $g(p^*_L) \geq g(p'_L)$ and that $g(p^*_L) = |\{i \in [n] : p^*_L(\overline{v}_i) = 1 + 3\theta\}|(\theta - \theta^2) + \theta^2 n + (2\theta - \theta^2)m$.*

Proof (Sketch). First we prove that, given $j \in [m]$ such that $p'_L(v_j^1) = p'_L(v_j^4) = 1 + 3\theta$, the leader's revenue does not decrease when decreasing the value $p'_L(v_j^1)$ to 4θ. By repeating this operation $\forall j \in [m]$ such that $p'_L(v_j^1) = p'_L(v_j^4) = 1 + 3\theta$, we obtain p_L^* which, by induction, satisfies $g(p_L^*) \geq g(p_L)$. Hence, we get $g(p_L^*, \overline{v}_i^1, \overline{v}_i^2) = (p_L^*(\overline{v}_i^1) - 3\theta)\theta \ \forall i \in [n]$ and $g(p_L^*, v_j^1, v_j^5) = 2\theta - \theta^2$. By summing these values for all drivers, we obtain the claimed value for $g(p_L^*)$. □

Now, we prove the theorem. Let $1 + \epsilon$ be a lower-bound on the approximability of MISP (see [1] for a proof of the APX-hardness of MISP). We prove, by contradiction, that SFPP cannot be $(1 + \delta)$-approximable $\forall \delta < \frac{\epsilon}{13}$. Suppose that there exists a $(1 + \delta)$-approximation algorithm for the SFPP with $\delta < \frac{\epsilon}{13}$. Let p_L^* be an optimal pricing function and p_L be the pricing function returned by the approximation algorithm. We have that $g(p_L^*)/g(p_L) \leq 1 + \delta$. By Lemma 2, we can suppose without loss of generality that p_L verifies $p_L(\overline{v}_i^1) \in \{4\theta, 1 + 3\theta\}$ $\forall i \in [n]$ and $p_L(v_j^1) = 4\theta$ or $p_L(v_j^4) = 4\theta$ $\forall j \in [m]$. In fact, if this is not the case, we can apply the polynomial time transformation outlined in the proof of Lemma 2 so as to obtain from p_L a pricing function p'_L verifying the assumption and such that $p_L^*/p'_L \leq p_L^*/p_L$. Clearly, again by Lemma 2 and the optimality of p_L^*, the same assumption can also be made for p_L^*.

Let $M(p_L) := \{u_i \in U : p_L(\overline{v}_i^1) = 1 + 3\theta\}$. From the assumptions on p_L^* and p_L, it follows that $M^* := M(p_L^*)$ and $M := M(p_L)$ are independent sets of (U, F). By the optimality of p_L^* and because of the characterization of $g(p_L^*)$ given in Lemma 2, M^* has to be a maximum independent set of (U, F). Let $\theta := n^{-2}$. Again by the characterization of the leader's revenue given in Lemma 2, for a sufficiently big n, we have

$$\frac{g(p_L^*)}{g(p_L)} = \frac{(|M^*| + 2m + (n - |M^*| - m)n^{-2})n^{-2}}{(|M| + 2m + (n - |M| - m)n^{-2})n^{-2}} \sim \frac{|M^*| + 2m}{|M| + 2m} \leq 1 + \delta. \quad (1)$$

By manipulating (1), we have that $|M^*|/|M| \leq 1 + \delta + 2m\delta/|M|$. In a graph of degree 3 and m edges, we can find in polynomial time an independent set with at least $\lceil m/6 \rceil$ nodes. So, we can suppose that $|M| \geq m/6$. Therefore, $|M^*|/|M| \leq 1 + \delta + 2m\delta/|M| \leq 1 + 13\delta < 1 + \epsilon \Rightarrow |M^*|/|M| < 1 + \epsilon$, thus contradicting the $1 + \epsilon$ lower bound on the approximability of the MISP. □

Note that, in the claim of Theorem 1, we required that $|L| = \Theta(n)$. However, if we consider instances of the SFPP such that $|L| = O(1)$, then the problem can be efficiently solved. We prove this fact by designing an algorithm, called SFPP-CL, which solves a polynomial number of linear systems in order to determine the best possible pricing function for the leader.

Let $K = \{(s_i, t_i) : i \in [k]\}$. For a pair of nodes (u, v), let $c_\infty(u, v)$ be the minimum cost paid by a driver to go from u to v without stopping at gas stations in L. Given a driver going from u to v, consider an optimal itinerary $B \in \mathcal{B}^*(p_L, u, v)$. There exists an itinerary D extracted from B, that we call *strong itinerary*, of the form $D = (u = v_1^1, v_2^1, \ldots v_{t(1)}^1 = v_1^C, v_1^2, v_2^2, \ldots v_{t(2)}^2 = v_2^C, \ldots\ldots v_{r-1}^C, v_1^r, v_2^r, \ldots v_{t(r)}^r = v)$, with $v_s^C \in C \setminus L \ \forall s \in [r-1]$ while all the other nodes belong to L, such that the cost c and the revenue g with respect to B can be computed as follows (set $p_z^s := p_L(v_z^s)$, $d_z^s := d(v_z^s, v_{z+1}^s)$ and $c_\infty(v_r^C, v_1^{r+1}) := 0$):

$$c(p_L, D) := \sum_{s=1}^{r} \sum_{z=1}^{t(s)-1} p_z^s \cdot d_z^s + c_\infty^s, \quad g(p_L, D) := \sum_{s=1}^{r} \sum_{z=1}^{t(s)-1} (p_z^s - e) \cdot d_z^s.$$

Set $cg(p_L, D) := [c(p_L, D), g(p_L, D)]$ and let $\mathcal{D}(u,v)$ be the set of strong itineraries starting at u and ending at v. It holds that $cg(p_L, u, v) = \min_{D \in \mathcal{D}(u,v)} cg(p_L, D)$. From this fact, it follows that an optimal pricing function p_L^* is an optimal solution to an LP problem $LP(\{D_{uv}\}_{(u,v)\in K})$, yielded by a set of optimal strong itineraries $\{D_{uv}\}_{(u,v)\in K}$, whose objective function to be maximized is $g(p_L)$ and the constraints are: *(i)* $c(p_L, D_{uv}) \leq c(p_L, D)$ for each $D \in \mathcal{D}(u,v)$, and *(ii)* $e \leq p_L(v) \leq p_C(v)$ for each $v \in L$. From the LP Theory, we know that there exist $|L|$ constraints in $LP(\{D_{uv}\}_{(u,v)\in K})$ defining a linear system of $|L|$ equations in $|L|$ variables whose unique solution is p_L^*. By evaluating the revenue yielded by all the pricing functions that solve all the possible linear systems and returning the one giving the highest leader's revenue, we obtain an optimal pricing function p_L^*.

We now describe algorithm SFPP-CL, which is based on a refinement of the ideas outlined above. SFPP-CL is divided into two phases: an *initializing phase*, in which several data structures are created and a *running phase*, in which SFPP-CL evaluates the leader's revenue yielded by several pricing functions obtained trough the data structures defined during the initializing phase. Let $K_U = \{u \in V : (u,v) \in K\}$ and $K_V = \{v \in V : (u,v) \in K\}$.

SFPP-CL: initializing phase

Step 1 Compute $c_\infty(u,v)$ $\forall u, v \in V$.

Step 2 $\forall u \in L$, $v \in L \cup K_V$, let $X_{uv} := [e = x_{uv}(0), x_{uv}(1), \ldots, x_{uv}(r_{uv}) = p_C(u)]$ be the vector given by the abscissa of all the vertices of the polyhedron $\mathcal{P}_{uv} := \bigcap_{z \in C \cup \{v\}} \{[x,y] : e \leq x \leq p_C(u), y \leq f_{uv}^z(x) := d(u,z) \cdot x + c_\infty(z,v)\}$ listed in increasing order (in order to compute X_{uv}, see [13]). Let $Z_{uv} := [z_{uv}(s)]_{s \in [r_{uv}]}$ be the vector such that $z_{uv}(s) = \arg\max_{z \in C \cup \{v\}} \{d(u,z) : [x_{uv}(s), f_{uv}^z(x_{uv}(s))]$ is a vertex of $\mathcal{P}_{uv}\}$ $\forall s \in [r_{uv}]$.

The usefulness of the vectors X_{uv} and Z_{uv} is that, if an optimal pricing function p_L^* verifies that $p_L^*(u) \in]x_{uv}(s-1), x_{uv}(s)]$ (set $x_{uv}(-1) = -\infty$), then the itinerary $D_{uv}^* := [u, z_{uv}(s), v]$ minimizes $cg(p_L^*, D)$ among all the itineraries of the form $D = [u, z, v]$ with $z \in C \cup \{v\}$. This observation will imply the correctness of the running phase of SFPP-CL. Before describing this phase, we present an algorithm that, given p_L, computes $g(p_L)$ and which will be used as a subroutine in the running phase (note that this algorithm can also be used to certify that the SFPP belongs to NP).

SFPP-CL-C algorithm (SFPP-CL-Certificate algorithm)

Step 1 Given $a \in L$ and $b \in L \cup K_V$, let s_{ab} be the smallest strictly positive integer such that $p_L(a) \leq x_{ab}(s_{ab})$ and, given $(u,v) \in K$, let $G_{uv} = (V_{uv}, E_{uv}, w_{uv})$ be a connected weighted graph that has an edge (a,b) with $w(a,b) = [c_\infty(a,b), 0]$ if $a \in \{u\} \setminus L$ and $b \in L \cup \{v\}$, $w(a,b) = [p_L(a) \cdot d(a, z_{ab}(s_{ab})) + c_\infty(z_{ab}(s_{ab}), b), (p_L(a) - e) \cdot d(a, z_{ab}(s_{ab}))]$ if $a \in L$ and $b \in L \cup \{v\}$.

Step 2 Evaluate the length of the shortest path from u to v in the graph G_{uv} $\forall (u,v) \in K$, which is equal to $g(p_L, u, v)$ (because of the observation we did when discussing the initializing phase). Finally, compute and return $g(p_L)$.

SFPP-CL**: running phase**

Step 1 Fix a set $K_L \subseteq K$ of $|L|$ elements. Let $K_U^L := K_U \cap K_L$ and $K_V^L := K_V \cap K_L$. Given $u \in L$, let $X_u = [e = x_u(0), x_u(1), \ldots, x_u(r_u) = p_C(u)]$ be the vector obtained by sorting the elements of the vectors X_{uv} with $v \in L \cup K_V^L$. Let $Z_u = (z_u(t,v))_{t \in [r_u], v \in L \cup K_V}$ be a matrix of nodes in V, in which the generic $z_u(t,v)$ is equal to the node $z_{uv}(s)$ if $x_{uv}(s-1) < x_u(t) \leq x_{uv}(s)$, where $s \in [r_{uv}]$.

Step 2 Let $T = [t(u)]_{u \in L}$ be a vector of integers indexed in L, such that $1 \leq t(u) \leq r_u$ $\forall u \in L$. Let $G_T = (V_T, E_T)$ be a connected graph such that E_T has the edges $(u, z_u(t(u), v)), (z_u(t(u), v), v), (z, v)$ with $z \in K_U^L$, $u \in L$, $v \in L \cup K_V^L$.

Step 3 Fix $L_-, L_+ \subseteq L$ such that $L_- \cap L_+ = \emptyset$ and set $r := |L| - |L_-| - |L_+|$. Let $\mathcal{D}_T^2 := \{(D_s, D_s')\}_{s \in [r]}$ be a set of r pairs of strong itineraries such that, $\forall s \in [r]$, D_s and D_s' both start at u and end at v, with $(u,v) \in K_L$ and are simple paths in G_T.

Step 4 Solve a linear system in the unknown variables yielded by p_L, with the following $|L|$ equations: $c(p_L, D_s) = c(p_L, D_s')$ $\forall s \in [r]$, $p_L(u) = x_u(t(u)-1)$ $\forall u \in L_-$ and $p_L(u) = x_u(t(u))$ $\forall u \in L_+$. If the system has only one solution p_L with $p_L(u) \in]x_u(t(u)-1), x_u(t(u))[$ $\forall u \in L \setminus \{L_- \cup L_+\}$, compute $g(p_L)$ by using SFPP-CL-C.

Step 5 Run Steps 1, 2, 3 and 4 for all possible choices of K_L, T, L_-, L_+ and $\mathcal{D}_T^2$ and return the best pricing function among the considered ones.

The following theorem holds.

Theorem 2 SFPP-CL *solves the* SFPP *in polynomial time when* $|L| = O(1)$.

3 An Approximation Algorithm: the Class SGPS

Given a generic pricing problem, a *uniform pricing function*, or UPF, is an assignment of the same price to all the priceable elements. An *optimal uniform pricing function*, or UPF*, maximizes the leader's revenue among all the UPFs. We introduce the class of *Stackelberg Games with Priceable Sets*, or SGPS, extending that of *Stackelberg Network Pricing Games* described in [6], and characterize the approximation factor provided by an UPF* in these games. Furthermore, we define an algorithm that finds an UPF^*, prove that the SFPP belongs to class SGPS and adapt such an algorithm to the SFPP.

A game in SGPS is a tuple $(E, C, L, p_C, w, \mathcal{E}, K, \{\lambda_i\}_{i\in K}, \{\mathcal{F}_i\}_{i\in K},)$ such that E is a set, $\{C, L\}$ is a partition of E, $p_C : C \to \mathbb{R}_{\geq 0}$ is a pricing function, $w : L \to \mathbb{R}_{\geq 0}$ is a weight function, $\mathcal{E} := \{E_1, E_2, \ldots E_r\}$ is a partition of L, K is a set of followers and, $\forall i \in K$, λ_i is the weight of follower i and $\mathcal{F}_i \subseteq \mathcal{P}(E)$ is the family of sets that can be purchased by follower i. We call *competitor's elements* the elements of C and *leader's elements* the elements of L. Given $p_L : \mathcal{E} \to \mathbb{R}_{\geq 0}$ and $F \subseteq E$, we define a cost c and a revenue g as follows:

$$g(p_L, F) := \sum_{E'\in\mathcal{E}} \sum_{e\in E'\cap F} w(e)\cdot p_L(E'), \quad c(p_L, F) := \sum_{e\in F\cap C} p_C(e) + g(p_L, F).$$

Let $cg(p_L, F) := [c(p_L, F), g(p_L, F)]$. Given $i \in K$, let $[c(p_L, i), g(p_L, i)] := cg(p_L, i) := \min_{F\in\mathcal{F}_i} cg(p_L, F)$, where the minimum is defined according to the ordering relation $\prec$. Informally, $c(p_L, i)$ is the cost of follower i when choosing a subset $F \in \mathcal{F}_i$ of minimum cost and $g(p_L, i)$ is the contribution of follower i to the leader's revenue recalling that, when a follower has different optimal choices, she adopts the one providing the highest revenue to the leader. Let $F^*(p_L, i) \in \mathcal{F}_i$ be a set minimizing the function $cg(p_L, i)$, that is, an optimal choice for follower i. The leader's revenue $g(p_L)$ is equal to $\sum_{i\in K} \lambda_i \cdot g(p_L, i)$. In order for g to be bounded, we suppose that, $\forall i \in K$, $\exists F \in \mathcal{F}_i$ such that $F \subseteq C$. Our objective is to find a pricing function p_L^* maximizing g. Different families of games in SGPS can be defined by different choices of $\{\mathcal{F}_i\}_{i\in K}$. The main differences with the Stackelberg Network Pricing Games are that single elements cannot be priced independently in general, since the leader has to assign the same price to every set in $\mathcal{E}$, and that the prices are weighted by the function w.

Given $\theta \geq 0$, p_θ is an UPF such that $p_\theta(E') = \theta\ \forall E' \in \mathcal{E}$. Let SG_1 be a game in SGPS with one follower only (it is then possible to avoid considering the follower's weight and to omit the index 1 in several functions) and let SG_2 be the game obtained from SG_1 by setting $\mathcal{E} = \{\{e\}\}_{e\in L}$. Note that the maximum leader's revenue g_1^* in SG_1 is lower than the maximum leader's revenue g_2^* in SG_2 and that a same UPF gives the same leader's revenues in both games. Hence, given the leader's revenues $g_{1,\theta}^*$ and $g_{2,\theta}^*$ yielded by an UPF^* for SG_1 and SG_2, respectively, we have that $g_1^*/g_{1,\theta}^* \leq g_2^*/g_{2,\theta}^*$. From this observation, in order to find un upper bound on the approximation factor yielded by an UPF^* for both games SG_1 and SG_2, we can restrict our analysis only to SG_2. Let $W, C : \mathcal{P}(E) \to \mathbb{R}_{\geq 0}$ be two functions such that $W(F) = \sum_{e\in F\cap L} w(e)\ \forall F \subseteq E$ and $C(F) = \sum_{e\in F\cap C} p_C(e)$. Let $X := \{W(F^*(p_\theta)) > 0 : \theta \geq 0\}$ be the *optimal weights* vector of follower 1 and consider it as a vector $X := [x(j) := x_j]_{j\in[n]}$ sorted in increasing order.

We define three functions $\theta, c, \varDelta : X \to \mathbb{R}_{\geq 0}$ such that, given $x_j \in X$, it holds that $\theta(x_j) := \theta_j := \max\{\theta \in \overline{\mathbb{R}}_{\geq 0} : W(F^*(p_\theta)) = x_j\}$, $c(x_j) := c_j := \min\{C(F) : W(F) = x_j, F \in \mathcal{F}\}$ and $\varDelta(x_j) := \varDelta_j := c(x_0) - c(x_j)$. Note that the function θ is decreasing in its argument. We call functions θ and c, respectively, the *optimal prices* function and the *optimal costs* function of follower 1. We observe that, $\forall j \in [n]$, assuming that $\theta_0 \cdot x_0 = \infty \cdot 0 = 0$, it holds that

$c(p_{\theta_j}) = c_j + \theta_j \cdot x_j = c_{j-1} + \theta_j \cdot x_{j-1}$. This implies the following equation:

$$\theta_j = \frac{c_{j-1} - c_j}{x_j - x_{j-1}} = \frac{\Delta_j - \Delta_{j-1}}{x_j - x_{j-1}}. \tag{2}$$

Let p_L^* be an optimal pricing function, the following relationship holds: $c(p_L^*) - g(p_L^*) = C(F^*(p_L^*)) \geq \min\{C(F) : F \in \mathcal{F}\} = C(F^*(p_0)) = c_n$. Moreover, we have $c(p_L^*) \leq c_0$. By using the previous inequalities, we get

$$g(p_L^*) = c(p_L^*) - (c(p_L^*) - g(p_L^*)) \leq c_0 - c_n = \Delta_n. \tag{3}$$

Let $W_m := x_1$, $W_M := x_n$, $\theta_m := \theta_n$, $\theta_M := \theta_1$. By exploiting inequalities (2) and (3), it is possible to prove the following fundamental theorem in analogy with the results in [6].

Theorem 3 *Any* UPF^* p_{θ^*} *provides an approximation guarantee of* $1 + \ln(\min\{W_M/W_m, \theta_M/\theta_m\})$ *to both* SG_1 *and* SG_2.

Now we can generalize the previous theorem to the case of more followers. We use the subscript i to denote the quantities previously defined for the case of a single follower when instantiated to follower i. Let SG_0 be a generic game in SGPS and define $\theta_m \leq \min_{i\in K} \theta_{m,i}$, $\theta_M \geq \max_{i\in K} \theta_{M,i}$, $W_m \leq \min_{i\in K} W_{m,i}$, $W_M \geq \max_{i\in K} W_{M,i}$. Let $\lambda_m := \min_{i\in K} \lambda_i$ and $\lambda_M := \sum_{i\in K} \lambda_i$.

The following result holds.

Theorem 4 *Any* UPF^* p_{θ^*} *provides an approximation guarantee of* $1 + \ln(\min\{(W_M/W_m) \cdot (\lambda_M/\lambda_m), \theta_M/\theta_m\})$ *to* SG_0.

Now, we design an algorithm to find an UPF^*. Given $i \in K$, let $X^i := [x^i(j) := x^i_j]_{j\in[n_i]}$ be the optimal weights vector of follower i and let $X := [x(h) := x_h]_{h\in[n]}$ be a vector sorted in increasing order containing all the values in X^i and such that $\exists F \in \mathcal{F}_i : x(h) = W(F)$ $\forall h \in [n]$, $\forall i \in K$. Moreover, let $C^i := [c^i(h) := c^i_h]_{h\in[n]}$ be the vector such that, $\forall h \in [n]$, $c^i(h) = c(x_h)$, where c denotes the optimal costs function of follower i. We suppose that the vectors X and $(C^i)_{i\in K}$ are known. Let $\Theta^i := [\theta^i(j) := \theta^i_j]_{j\in[n_i]}$ be the vector such that, $\forall j \in [n_i]$, $\theta^i(j) = \theta(x^i_j)$, where θ denotes the optimal prices function of follower i. Observe that Θ^i is the vector given by the positive abscissa of all the vertices of the polyhedron $\mathcal{P}_i := \bigcap_{h\in[n]} \left\{[\theta, y] : \theta \in \mathbb{R}, y \leq f^i_h(\theta) := c^i_h + \theta \cdot x_h\right\}$, and X^i is the vector such that $x^i_j = \max\{x_h : [\theta^i_j, f^i_h(\theta^i_j)]$ is a vertex of $\mathcal{P}_i\}$. By these characterizations we can compute $(X^i)_{i\in K}$ and $(\Theta^i)_{i\in K}$ in $O(|K|n\log(n))$ time (see [13] for further details). Let $\Theta = [\theta(h) := \theta_h]_{h\in[m]}$ be the sorted fusion of the vectors $(\Theta^i)_{i\in K}$ without the eventual null element. Θ can be computed in $O(|K|n\log(|K|))$ time. Given $i \in K$ and $h \in [m]$, let $j(h,i) \in [n_i]$ be such that $\theta^i_{j(h,i)} \geq \theta_h > \theta^i_{j(h,i)+1}$ (set $\theta^i_{n_1+1} := 0$). Observe that $W(F^*(p_{\theta_h}, i)) = x^i_{j(h,i)}$ and then $g(p_{\theta_h}, i) = \theta_h \cdot x^i_{j(h,i)}$. Since there exists $h^* \in [m]$ such that $p_{\theta_{h^*}}$ is a UPF^*, by the above arguments, we have that

$$g(p_{\theta_{h^*}}) = \max_{h\in[m]} g(p_{\theta_h}) = \max_{h\in[m]} \sum_{i\in K} \lambda_i \cdot g(p_{\theta_h}, i) = \max_{h\in[m]} \sum_{i\in K} \lambda_i \cdot \theta_h \cdot x^i_{j(h,i)}. \tag{4}$$

Observe that $g(p_{\theta_{h^*}})$ can be computed in $O(|K|n)$ time, by using the characterization provided in (4). So, we can define an algorithm that returns a UPF^* starting from X and $(C^i)_{i\in K}$, and that runs in $O(|K|n(\log(|K|)+\log(n)))$ time. We call this algorithm *Perfect-Single-Price* algorithm (PSP); it differs from the *Single-Price* algorithm given in [6] since the approximation ratio of the latter (which is worse than the one of PSP algorithm) and its complexity depend on an arbitrary ϵ given in input. An approximation ratio provided by this algorithm can be obtain by setting $W_m := x(1)$, $W_M := x(m)$, $\theta_m := \theta(m)$, $\theta_M := \theta(1)$.

To apply the PSP algorithm to the SFPP, we reformulate the SFPP as a game in SGPS. Given an instance $I := (G, K, \{\lambda_{uv}\}_{(u,v)\in K}, L, C, p_C, e)$ of the SFPP, define an instance $I' := (E', C', L', p'_C, w', \mathcal{E}, K, \{\lambda_{uv}\}_{(u,v)\in K}, \{\mathcal{F}_{uv}\}_{(u,v)\in K})$ in the class SGPS as follows. Let $E' = C' \cup L'$ be a set of edges in a graph (V', E') defined as follows: given $u \in L$, $v \in C$ and $z \in V$, create a node z'_{uv}, insert in L' an edge (u, z'_{uz}) with $w'((u, z'_{uz})) = d(u, z)$, insert in C' an edge (z'_{uz}, z) with $p'_C((z'_{uz}, z)) = e \cdot d(u, z)$, insert in C' an edge (v, z) with $p'_C((v, z)) = p_C(v) \cdot d(v, z)$. Given $u \in L$, set $E_u = \{(u, z'_{uz}) \in L'\}$ and $\mathcal{E} := \{E_u\}_{u\in L}$. Set, $\forall (u, v) \in K$, $\mathcal{F}_{uv}$ equal to the set of all the paths connecting u to v in (V', E'). Given two pricing functions p_L and p'_L for the problems I and I', respectively, such that $p_L(u) = p'_L(E_u) + e\ \forall u \in L$, we have that $g(p_L) = g(p'_L)$. Moreover, by assigning uniformly the prices θ and $\theta' := \theta - e$ to I and I', respectively, we obtain that the length of the sub-path covered by follower $(u, v) \in K$ by using fuel bought from the leader in I, is equal to $W(F^*_{uv}(p_{\theta'}))$. From these observations, if one transforms I into I' and applies the PSP algorithm to I', I can be approximated according to the performance guarantee stated in Theorem 4.

To apply the PSP algorithm, we need a preliminary procedure to find the vectors X and $(C^{uv})_{(u,v)\in K}$. Observe that we can set $X := \{d(u, v) > 0 : u \in L, v \in C \cup K_V\} := [x(h)]_{h\in[l]}$ to apply the PSP algorithm, since every follower, given an UPF, can always choose a path such that only a subpath is covered using fuel bought from the leader. Because of this fact, we also have that $C^{uv}(h) = \min\{c_\infty(u, z) + e \cdot d(z, z') + c_\infty(z', v) : d(z, z') = x(h), z \in L, z' \in C \cup \{v\}\}$ $\forall h \in [l]$ and $\forall (u, v) \in K$. Observe that, given $(u, v) \in K$, C_{uv} can be computed in $O(|L| \cdot |C|) \subseteq O(n^2)$ time. To approximate the SFPP, compute the vectors X and $(C_{uv})_{(u,v)\in K}$ and apply the PSP algorithm to these vectors. The resultant algorithm, that we call SFPP-PSP, requires $O(n^4)$ time in the worst-case.

We conclude by showing that the approximation ratio provided by the SFPP-PSP algorithm is tight and also significantly high. Consider an instance with $V = \{v_i\}_{i\in[n+1]}, E = \{(v_i, v_{i+1})\}_{i\in[n]}$, $C = L = V \setminus \{v_{n+1}\}$, $w((v_i, v_{i+1})) = 2^{i-n}$ and $p_C(v_i) = 2^{n-i}\ \forall i \in [n]$, $e = 0$, $K = \{(v_1, v_{n+1})\}$. Observe that the optimal assignment p^*_L verifies that $p^*_L(v_i) = p_C(v_i)\ \forall i \in [n-1]$ and so $g(p^*_L) = \sum_{i=1}^n 2^{i-n} \cdot 2^{n-i} = n$. Observe that an UPF^* is of the form $p_{(2^i)}$, with $i \in [n-1] \cup \{0\}$, and we have that $g(p_{(2^i)}) = \sum_{j=0}^{n-1-i} 2^{-j}$. Therefore $p_{(2^0)} = p_{(1)}$ is an UPF^* and $g(p_{(1)}) = \sum_{j=0}^{n-1} 2^{-j} \overset{n\to\infty}{\longrightarrow} 2$. Hence, the approximation ratio provided by the SFPP-PSP algorithm is asymptotically equal to $g(p^*)/g(p_{(1)}) = n/2$, and so it is very high. Since this instance verifies $1 + \ln(W_M/W_m) = 1 + \ln(\sum_{i=1}^n w((v_i, v_{i+1}))/w((v_1, v_2))) = 1 + \ln(\sum_{i=0}^{n-1} 2^i) = 1 + \ln(2^n - 1) \sim \ln(2^n) =$

$\ln(4) \cdot n/2 \simeq 1.386 \cdot n/2$ and $1 + \ln(\theta_M/\theta_m) = 1 + \ln(2^{n-1}) \sim \ln(4) \cdot n/2$, we have that the approximation ratio claimed in Theorem 3 for the SFPP may not be asymptotically improved. By using an instance of SFPP similar to the previous one, it is possible to prove that also the approximation ratio $1 + \ln(\lambda_M/\lambda_m)$ claimed in Theorem 4 is asymptotically tight.

References

1. P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Proc. of the 3rd Italian Conference on Algorithms and Complexity (CIAC)*, LNCS 1203, Springer, pp 288–298, 1997.
2. D. Bilò, L. Gualà, S. Leucci, and G. Proietti. Specializations and generalizations of the Stackelberg minimum spanning tree game. In *Proc. of the 6th Workshop on Internet and Network Economics (WINE)*, LNCS 6484, Springer, pp. 75–86, 2010.
3. D. Bilò, L. Gualà, G. Proietti, and P. Widmayer. Computational aspects of a 2-player Stackelberg shortest paths tree game. In *Proc. of the 4th Workshop on Internet and Network Economics (WINE)*, LNCS 4858, Springer, pp. 251–262, 2008.
4. M. Bouhtou, A. Grigoriev, S. van Hoesel, A. van der Kraaij, F. Spieksma, and M. Uetz. Pricing bridges to cross a river. *Naval Research Logistics*, 54(4):411–420, 2007.
5. P. Briest, P. Chalermsook, S. Khanna, B. Laekhanukit, and D. Nanongkai. Improved hardness of approximation for Stackelberg shortest-path pricing. In *Proc. of the 6th Workshop on Internet and Network Economics (WINE)*, LNCS 6484, Springer, pp. 444–454, 2010.
6. P. Briest, M. Hoefer, and P. Krysta. Stackelberg network pricing games. *Algorithmica*, 62(3-4):733–753, 2012.
7. J. Cardinal, E. D. Demaine, S. Fiorini, G. Joret, S. Langerman, I. Newman, and O. Weimann. The Stackelberg minimum spanning tree game. *Algorithmica*, 59(2):129–144, 2011.
8. J. Cardinal, E. D. Demaine, S. Fiorini, G. Joret, I. Newman, and O. Weimann. The Stackelberg minimum spanning tree game on planar and bounded-treewidth graphs. *Journal of Combinatorial Optimization*, 25(1):19–46, 2013.
9. S. Khuller, A. Malekian, and J. Mestre. To fill or not to fill: The gas station problem. *ACM Transactions on Algorithms*, 7(3):36, 2011.
10. D. G. Kirkpatrick, and R. Seidel. The ultimate planar convex hull algorithm. *SIAM Journal on Computing*, 15(1):287–299, 1986.
11. M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44(12):1608–1622, 1998.
12. S. Roch, G. Savard, and P. Marcotte. An approximation algorithm for Stackelberg network pricing. *Networks*, 46(1):57-67, 2005.
13. M. I. Shamos, and D. Hoey. Geometric intersection problems. In *Proc. of the 17th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, pp. 208–215, 1976.
14. S. van Hoesel. An overview of Stackelberg pricing in networks. Research Memoranda 042, Maastricht: METEOR, Maastricht Research School of Economics of Technology and Organization, 2006.
15. H. von Stackelberg. *Marktform und Gleichgewicht (Market and Equilibrium)*. Verlag von Julius Springer, Vienna, 1934.

Structural complexity of multi-valued partial functions computed by nondeterministic pushdown automata
(Extended Abstract)

Tomoyuki Yamakami

Department of Information Science, University of Fukui
3-9-1 Bunkyo, Fukui 910-8507, Japan

Abstract. This paper continues a systematic and comprehensive study on the structural properties of CFL functions, which are in general multi-valued partial functions computed by one-way one-head nondeterministic pushdown automata equipped with write-only output tapes (or pushdown transducers), where CFL refers to a relevance to context-free languages. The CFL functions tend to behave quite differently from their corresponding context-free languages. We extensively discuss containments, separations, and refinements among various classes of functions obtained from the CFL functions by applying Boolean operations, functional composition, many-one relativization, and Turing relativization. In particular, Turing relativization helps construct a hierarchy over the class of CFL functions. We also analyze the computational complexity of optimization functions, which are to find optimal values of CFL functions, and discuss their relationships to the associated languages.

Keywords: multi-valued partial function, oracle, Boolean operation, refinement, many-one relativization, Turing relativization, CFL hierarchy, optimization, pushdown automaton, context-free language

1 Much Ado about Functions

In a traditional field of formal languages and automata, we have dealt primarily with *languages* because of their practical applications to, for example, a parsing analysis of programming languages. Most fundamental languages listed in many undergraduate textbooks are, unarguably, regular (or rational) languages and context-free (or algebraic) languages.

Opposed to the recognition of languages, translation of words, for example, requires a mechanism of transforming input words to output words. Aho et al. [1] studied machines that produce words on output tapes while reading symbols on an input tape. Mappings on strings (or word relations) that are realized by such machines are known as transductions. Since languages are regarded, from an integrated viewpoint, as Boolean-valued (i.e., $\{0,1\}$-valued) total functions, it seems more essential to study the behaviors of those functions. This task is, however, quite challenging, because these functions often demand quite different concepts, technical tools, and proof arguments, compared to those for languages. When underlying machines are particularly *nondeterministic*, they may produce numerous distinct output values (including the case of no output values). Mappings realized by such machines become, in general, *multi-valued partial functions* transforming each admissible string to a certain finite (possibly empty) set of strings.

Based on a model of polynomial-time nondeterministic Turing machine, computational complexity theory has long discussed the structural complexity of various NP function classes, including NPMV, NPSV, and $\mathrm{NPSV_t}$ (where MV and SV respectively stand for "multi-valued" and "single-valued" and the subscript "t" does for "total"). See, e.g., a survey [14].

Within a scope of formal languages and automata, there is also rich literature concerning the behaviors of nondeterministic finite automata equipped with write-only output tapes (known as rational transducers) and properties of associated multi-valued partial functions (known also as rational transductions). Significant efforts have been made over the years to understand the functionality of such functions. A well-known field of functions include "CFL functions," which were formally discussed in 1963 by Evey [4] and Fisher [6] and general properties have been since then discussed in, e.g., [3, 10]. CFL functions are generally computed by *one-way one-head nondeterministic pushdown automata* (or *npda's*, in short) *equipped with write-only output tapes*. For example, the function $PAL_{sub}(w) = \{x \in \{0,1\}^* \mid \exists u, v\, [w = uxv], x = x^R\}$ for every $w \in \{0,1\}^*$ is a CFL function, where x^R is x in reverse. As subclasses of CFL functions, three fundamental function classes CFLMV, CFLSV, and $\mathrm{CFLSV_t}$ were recognized explicitly in [19] and further explored in [20].

In recent literature, fascinating structural properties of CFL functions have been extensively investigated. Konstantinidis et al. [10] took an algebraic approach toward the characterization of CFL functions. In relation to cryptography, it was shown that there exists a pseudorandom generator in $\mathrm{CFLSV_t}$ that "fools" every language in a non-uniform (or an advised) version of REG [19]. Another function class CFLMV(2) in [20] contains pseudorandom generators against a non-uniform analogue of CFL. The behaviors of functions in those function classes seem to look quite different from what we have known for context-free languages. For instance, the single-valued total function class CFLSV can be seen as a functional extension of the language family CFL $\cap$ co-CFL rather than CFL [20]. In stark contrast with a tidy theory of NP functions, circumstances that surround CFL functions differ significantly because of mechanical constraints (e.g., a use of stacks, one-way moves of tape heads, and λ-moves) that harness the behaviors of underlying npda's with output tapes. One such example is concerning a notion of *refinement* [13] (or uniformization [10]). Unlike language families, a containment between two multi-valued partial functions is customarily replaced by refinement. Konstantinidis et al. [10] posed a basic question of whether every function in CFLMV has a refinement in CFLSV. This question was lately solved negatively [22] and this result clearly contrasts a situation that a similar relationship is not known to hold between NPMV and NPSV.

Amazingly, there still remains a vast range of structural properties that await for further investigation. Thus, we wish to continue a coherent and systematic study on the structural behaviors of the aforementioned CFL functions. This paper aims at exploring fundamental relationships (such as, containment, separation, and refinement) among the above function classes and their natural extensions via four typical operations: (i) Boolean operations, (ii) functional composition, (iii) many-one relativization, and (iv) Turing relativization. The last two operations are a natural generalization of *many-one and Turing CFL-reducibilities* among languages [21]. We use the Turing relativization to introduce a hierarchy of function classes $\Sigma_k^{\mathrm{CFL}}\mathrm{MV}$, $\Pi_k^{\mathrm{CFL}}\mathrm{MV}$, and $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ for each level $k \geq 1$, in which the first Σ-level classes coincide with CFLMV and CFLSV, respectively. We show that all functions in this hierarchy have linear space-complexity. With regard to refinement, we show that, if the CFL hierarchy of [21] collapses to the kth level, every function in $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV}$ has a refinement in $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$ for every index $k \geq 2$.

Every nondeterministic machine with an output tape can be naturally seen as a process of generating a set of feasible "solutions" of each instance. Among those solutions, it is useful to study the complexity of finding "optimal" solutions. This gives rise to *optimization functions*. Earlier, Krentel [11] discussed properties of OptP that is composed of polynomial-time optimization functions. Here, we are focused on similar functions induced by npda's with output tapes. We denote by OptCFL a class of those optimization CFL functions. This function class is proven to be located between $\mathrm{CFLSV_t}$ and $\Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$. Moreover, we show the class separation between $\mathrm{CFLSV_t}$ and OptCFL.

To see a role of functions during a process of recognizing languages, Köbler and Thierauf [9] introduced a *//-advice operator* by generalizing the Karp-Lipton /-advice operator, and they argued the computational complexity of languages in $\mathrm{P}//\mathcal{F}$ and $\mathrm{NP}//\mathcal{F}$ induced by any given function class $\mathcal{F}$. Likewise, we discuss the complexity of $\mathrm{REG}//\mathcal{F}$ and $\mathrm{CFL}//\mathcal{F}$ when $\mathcal{F}$ are various subclasses of CFL functions, particularly $\mathrm{CFLSV_t}$ and OptCFL.

All omitted or abridged proofs, because of the page limit, will appear shortly in a complete version of this paper.

2 A Starting Point

Formal Languages. Let $\mathbb{N}$ be the set of all *natural numbers* (i.e., nonnegative integers) and set $\mathbb{N}^+ = \mathbb{N} - \{0\}$. Throughout this paper, we use the term " polynomial" to mean polynomials on $\mathbb{N}$ with nonnegative coefficients. In particular, a *linear polynomial* is of the form $ax+b$ with $a, b \in \mathbb{N}$. The notation $A - B$ for two sets A and B indicates the *set difference* $\{x \mid x \in A, x \notin B\}$. Given any set A, $\mathcal{P}(A)$ denotes the *power set* of A. A set Σ^k (resp., $\Sigma^{\leq k}$), where $k \in \mathbb{N}$, consists only of strings of length k (resp., at most k). Here, we set $\Sigma^* = \bigcup_{k \in \mathbb{N}} \Sigma^k$. The *empty string* is always denoted λ. Given any language A over Σ, its *complement* is $\Sigma^* - A$, which is also denoted by $\overline{A}$ as long as Σ is clear from the context.

We adopt a *track notation* $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$ from [15]. For two symbols σ and τ, $[\begin{smallmatrix} \sigma \\ \tau \end{smallmatrix}]$ expresses a new symbol. For two strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_n$ of length n, $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$ denotes a string $[\begin{smallmatrix} x_1 \\ y_1 \end{smallmatrix}][\begin{smallmatrix} x_2 \\ y_2 \end{smallmatrix}]\cdots[\begin{smallmatrix} x_n \\ y_n \end{smallmatrix}]$. Whenever $|x| \neq |y|$, we follow a convention introduced in [15]: if $|x| < |y|$, then $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$ actually means $[\begin{smallmatrix} x\#^m \\ y \end{smallmatrix}]$, where $m = |y| - |x|$ and $\#$ is a designated new symbol. Similarly, when $|x| > |y|$, the notation $[\begin{smallmatrix} x \\ y \end{smallmatrix}]$ expresses $[\begin{smallmatrix} x \\ y\#^m \end{smallmatrix}]$ with $m = |x| - |y|$.

As our basic computation model, we use *one-way one-head nondeterministic pushdown automata* (or npda's, in short) allowing λ-moves (or λ-transitions) of their input-tape heads. The notations REG and CFL stand for the families of all regular languages and of all context-free languages, respectively. For each number $k \in \mathbb{N}^+$, the *k-conjunctive closure* of CFL, denoted by CFL(k), is defined to be $\{\bigcap_{i=1}^{k} A_i \mid A_1, A_2, \ldots, A_k \in \mathrm{CFL}\}$ (see, e.g., [19]).

Given any language A (used as an oracle), CFL_T^A (or $\mathrm{CFL}_T(A)$) expresses a collection of all languages recognized by npda's equipped with write-only query tapes with which the npda's make non-adaptive oracle queries to A, provided that all computation paths of the npda's must terminate in $O(n)$ steps no matter what oracle is used [21]. We use the notation $\mathrm{CFL}_T^{\mathcal{C}}$ (or $\mathrm{CFL}_T(\mathcal{C})$) for language family $\mathcal{C}$ to denote the union $\bigcup_{A\in\mathcal{C}} \mathrm{CFL}_T^A$. Its deterministic version is expressed as $\mathrm{DCFL}_T^{\mathcal{C}}$. The *CFL hierarchy* $\{\Delta_k^{\mathrm{CFL}}, \Sigma_k^{\mathrm{CFL}}, \Pi_k^{\mathrm{CFL}} \mid k \in \mathbb{N}^+\}$ is composed of classes $\Delta_1^{\mathrm{CFL}} = \mathrm{DCFL}$, $\Sigma_1^{\mathrm{CFL}} = \mathrm{CFL}$, $\Pi_k^{\mathrm{CFL}} = \text{co-}\Sigma_k^{\mathrm{CFL}}$, $\Delta_{k+1}^{\mathrm{CFL}} = \mathrm{DCFL}_T(\Pi_k^{\mathrm{CFL}})$, and $\Sigma_{k+1}^{\mathrm{CFL}} = \mathrm{CFL}_T(\Pi_k^{\mathrm{CFL}})$ for each index $k \geq 1$ [21].

Functions and Refinement. Our terminology associated with *multi-valued partial functions* closely follows the standard terminology in computational complexity theory. Throughout this paper, we adopt the following convention: the generic term "function" always refers to "multi-valued partial function," provided that *single-valued* functions are viewed as a special case of multi-valued functions and, moreover, partial functions include *total* functions. We are interested in multi-valued partial functions mapping[1] Σ^* to $\mathcal{P}(\Gamma^*)$ for certain alphabets Σ and Γ. When f is single-valued, we often write $f(x) = y$ instead of $y \in f(x)$. Associated with f, dom(f) denotes the *domain* of f, defined to be $\{x \in \Sigma^* \mid f(x) \neq \emptyset\}$. If $x \notin \mathrm{dom}(x)$, then $f(x)$ is said to be *undefined.* The *range* ran(f) of f is a set $\{y \in \Gamma^* \mid f^{-1}(y) \neq \emptyset\}$.

For any language A, the *characteristic function* for A, denoted by χ_A, is a function defined as $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ otherwise. We also use a *quasi-characteristic function* η_A, which is defined as $\eta_A(x) = 1$ for any string x in A and $\eta_A(x)$ is not defined for all the other strings x.

Concerning all function classes discussed in this paper, it is natural to concentrate only on functions whose outcomes are bounded in size by fixed polynomials. More precisely, a multi-valued partial function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is called *polynomially bounded* (resp., *linearly bounded*) if there exists a polynomial (resp., a linear polynomial) p such that, for any two strings $x \in \mathrm{dom}(f)$ and $y \in \Gamma^*$, if

[1] To describe a multi-valued partial function f, the expression "$f : \Sigma^* \to \Gamma^*$" is customarily used in the literature. However, the current expression "$f : \Sigma^* \to \mathcal{P}(\Gamma^*)$" matches a fact that the outcome of f on each input string in Σ^* is a subset of Γ^*.

$y \in f(x)$ then $|y| \leq p(|x|)$ holds. In this paper, we understand that *all function classes are made of polynomially-bounded functions.* Given two alphabets Σ and Γ, a function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is called *length preserving* if, for any $x \in \Sigma^*$ and $y \in \Gamma^*$, $y \in f(x)$ implies $|x| = |y|$.

Whenever we refer to a *write-only tape*, we always assume that (i) initially, all cells of the tape are blank, (ii) a tape head starts at the so-called *start cell*, (iii) the tape head steps forward whenever it writes down any non-blank symbol, and (iv) the tape head can stay still only in a blank cell. An *output* (outcome or output string) along a computation path is a string produced on the output tape after the computation path is terminated. We call an output string *valid* (or *legitimate*) if it is produced along a certain accepting computation path.

To describe npda's, we take the following specific definition. For any given function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$, an npda N equipped with a one-way read-only input tape and a write-only output tape that computes f must have the form $(Q, \Sigma, \{¢, \$\}, \Theta, \Gamma, \delta, q_0, Z_0, Q_{acc}, Q_{rej})$, where Q is a set of inner states, Θ is a stack alphabet, q_0 ($\in Q$) is the initial state, Z_0 ($\in \Theta$) is the stack's bottom marker, and $\delta : (Q - Q_{halt}) \times (\check{\Sigma} \cup \{\lambda\}) \times \Theta \to \mathcal{P}(Q \times \Theta^* \times (\Gamma \cup \{\lambda\}))$ is a transition function, where $Q_{halt} = Q_{acc} \cup Q_{rej}$, $\check{\Sigma} = \Sigma \cup \{¢, \$\}$, and ¢ and \$ are respectively left and right endmarkers. It is important to note that, in accordance with the basic setting of [21], we consider only *npda's whose computation paths are all terminate (i.e., reach halting states) in $O(n)$ steps*,[2] where n refers to input size. We refer to this specific condition regarding execution time as the *termination condition.*

A function class CFLMV is composed of all (multi-valued partial) functions f, each of which maps Σ^* to $\mathcal{P}(\Gamma^*)$ for certain alphabets Σ and Γ and there exists an npda N with a one-way read-only input tape and a write-only output tape such that, for every input $x \in \Sigma^*$, $f(x)$ is a set of all *valid* outcomes of N on the input x. The termination condition imposed on our npda's obviously leads to an anticipated containment CFLMV $\subseteq$ NPMV. Another class CFLSV is a subclass of CFLMV consisting of single-valued partial functions. In addition, $\mathrm{CFLMV_t}$ and $\mathrm{CFLSV_t}$ are respectively restrictions of CFLMV and CFLSV onto *total functions.* Those function classes were discussed earlier in [19].

An important concept associated with multi-valued partial functions is *refinement* [13]. This concept (denoted by $\sqsubseteq_{ref}$) is more suitable to use than *set containment* ($\subseteq$). Given two multi-valued partial functions f and g, we say that f is a *refinement* of g, denoted by $g \sqsubseteq_{ref} f$, if (1) $\mathrm{dom}(f) = \mathrm{dom}(g)$ and (2) for every x, $f(x) \subseteq g(x)$ (as a set inclusion). We also say that g is *refined* by f. Given two sets $\mathcal{F}$ and $\mathcal{G}$ of functions, $\mathcal{G} \sqsubseteq_{ref} \mathcal{F}$ if every function g in $\mathcal{G}$ can be refined by a certain function f in $\mathcal{F}$. When f is additionally single-valued, we call f a *single-valued refinement* of g.

[2] If no execution time bound is imposed, a function computed by an npda that nondeterministically produces every binary string on its output tape for each input becomes a valid CFL function; however, this function is no longer an NP function.

3 Basic Operations for Function Classes

Let us discuss our theme of the structural complexity of various classes of (multi-valued partial) functions by exploring fundamental relationships among those function classes. In the course of our discussion, we will unearth an exquisite nature of CFL functions, which looks quite different from that of NP functions.

We begin with demonstrating basic features of CFL functions. First, let us present close relationships between CFL functions and context-free languages. Notice that, for any function f in CFLMV, both $\mathrm{dom}(f)$ and $\mathrm{ran}(f)$ belong to CFL. It is useful to recall from [21] a notion of $\natural$-extension. Assuming that $\natural \notin \Sigma$, a *$\natural$-extension* of a given string $x \in \Sigma^*$ is a string $\tilde{x}$ over $\Sigma \cup \{\natural\}$ satisfying the following requirement: x is obtained directly from $\tilde{x}$ simply by removing all occurrences of $\natural$ in $\tilde{x}$. For example, if $x = 01101$, then $\tilde{x}$ may be $01\natural1\natural01$ or $\natural0110\natural\natural1$. The next lemma resembles Nivat's representation theorem for rational transductions (see, e.g., [10, Theorem 2]).

Lemma 1. *For any function $f \in$ CFLMV, there exist a language $A \in$ CFL and a linear polynomial p such that, for every pair x and y, $y \in f(x)$ iff (i) $[\,{\tilde{x} \atop \tilde{y}}\,] \in A$, (ii) $|y| \leq p(|x|)$, and (iii) $|\tilde{x}| = |\tilde{y}|$ for certain strings $\tilde{x}$ and $\tilde{y}$, which are $\natural$-extensions of x and y, respectively.*

An immediate application of Lemma 1 leads to a functional version of the well-known pumping lemma [2].

Lemma 2. (functional pumping lemma for CFLMV) *Let Σ and Γ be any two alphabets and let $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ be any function in* CFLMV*. There exist three numbers $m \in \mathbb{N}^+$ and $c, d \in \mathbb{N}$ that satisfy the following condition. Any string $w \in \Sigma^*$ with $|w| \geq m$ and any string $s \in f(w)$ are decomposed into $w = uvxyz$ and $s = abpqr$ such that (i) $|vxy| \leq m$, (ii) $|vybq| \geq 1$, (iii) $|bq| \leq cm + d$, and (iv) $ab^ipq^ir \in f(uv^ixy^iz)$ for any number $i \in \mathbb{N}$. In the case where f is further length preserving, the following condition also holds: (v) $|v| = |b|$ and $|y| = |q|$. Moreover, (i)–(ii) can be replaced by (i') $|bq| \geq 1$.*

Boolean operations over languages in CFL have been extensively discussed in the past literature (e.g., [16, 21]). Similarly, it is possible to consider Boolean operations that are directly applicable to functions. In particular, we are focused on three typical Boolean operations: union, intersection, and complement. Let us define the first two operations. Given two function classes $\mathcal{F}_1$ and $\mathcal{F}_2$, let $\mathcal{F}_1 \wedge \mathcal{F}_2$ (resp., $\mathcal{F}_1 \vee \mathcal{F}_2$) denote a class of all functions f defined as $f(x) = f_1(x) \cap f_2(x)$ (set intersection) (resp., $f(x) = f_1(x) \cup f_2(x)$, set union) over all inputs x, where $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$. Expanding CFLMV(2) in Section 1, we inductively define a *k-conjunctive function class* CFLMV(k) as follows: CFLMV(1) = CFLMV and CFLMV($k+1$) = CFLMV(k) $\wedge$ CFLMV for any index $k \in \mathbb{N}^+$. Likewise, CFLSV(k) is defined using CFLSV instead of CFLMV.

Proposition 3. *Let $k, m \geq 1$.*

1. CFLMV($\max\{k, m\}$) $\subseteq$ CFLMV(k) $\vee$ CFLMV(m) $\subseteq$ CFLMV(km).

2. $\mathrm{CFLMV}(\max\{k,m\}) \subseteq \mathrm{CFLMV}(k) \wedge \mathrm{CFLMV}(m) \subseteq \mathrm{CFLMV}(k+m)$.
3. $\mathrm{CFLSV}(k) \subsetneqq \mathrm{CFLSV}(k+1)$.

Note that Proposition 3(3) follows indirectly from a result in [12].

Fenner et al. [5] considered "complements" of NP functions. Likewise, we can discuss *complements of CFL functions*. Let $\mathcal{F}$ be any family of functions whose output sizes are upper-bounded by certain *linear polynomials*. A function f is in co-$\mathcal{F}$ if there are a linear polynomial p, another function $g \in \mathcal{F}$, and a constant $n_0 \in \mathbb{N}$ such that, for any pair (x,y) with $|x| \geq n_0$, $y \in f(x)$ iff both $|y| \leq p(|x|)$ and $y \notin g(x)$ holds. This condition implies that $f(x) = \Sigma^{\leq a|x|+b} - g(x)$ (set difference) for all $x \in \Sigma^{\geq n_0}$. The finite portion $\Sigma^{<n_0}$ of inputs is ignored.

The use of set difference in the above definition makes us introduce another class operator $\ominus$. Given two sets $\mathcal{F},\mathcal{G}$ of functions, $\mathcal{F} \ominus \mathcal{G}$ denotes a collection of functions h satisfying the following: for certain two functions $f \in \mathcal{F}$ and $g \in \mathcal{G}$, $h(x) = f(x) - g(x)$ (set difference) holds for any x.

In Proposition 4, we will give basic properties of functions in co-CFLMV and of the operator $\ominus$. To describe the proposition, we need to introduce a new function class, denoted by NFAMV, which is defined in a similar way of introducing CFLMV using, in place of npda's, *one-way (one-head) nondeterministic finite automata* (or nfa's, in short) with write-only output tapes, provided that the termination condition (i.e., all computation paths terminate in linear time) must hold.

Proposition 4. *1.* co-(co-CFLMV) = CFLMV.
2. co-CFLMV = NFAMV $\ominus$ CFLMV.
3. CFLMV $\ominus$ CFLMV = CFLMV $\wedge$ co-CFLMV.
4. CFLMV $\neq$ co-CFLMV. *The same holds for* $\mathrm{CFLMV_t}$.

Proof Sketch. We will show only (2). ($\supseteq$) Let $f \in$ NFAMV $\ominus$ CFLMV and take two functions $h \in$ NFAMV and $g \in$ CFLMV for which $f(x) = h(x) - g(x)$ (set difference) for all inputs $x \in \Sigma^*$. Choose a linear polynomial p satisfying that, for every pair (x,y), $y \in h(x) \cup g(x)$ implies $|y| \leq p(|x|)$. By the definition of f, it holds that $f(x) = \Sigma^{\leq p(|x|)} - (g(x) \cup (\Sigma^{\leq p(|x|)} - h(x)))$ for all x. For simplicity, we define $r(x) = g(x) \cup (\Sigma^{\leq p(|x|)} - h(x))$ for every x. It thus holds that $f(x) = \Sigma^{\leq p(|x|)} - r(x)$. It is not difficult to show that r is in CFLMV.

($\subseteq$) Let $f \in$ co-CFLMV. There are a linear polynomial p and a function $g \in$ CFLMV satisfying that $f(x) = \Sigma^{\leq p(|x|)} - g(x)$ for all x. We set $h(x) = \Sigma^{\leq p(|x|)}$. Since $h \in$ NFAMV, we conclude that f belongs to NFAMV $\ominus$ CFLMV. □

Another basic operation used for functions is *functional composition*. The functional composition $f \circ g$ of two functions f and g is defined as $(f \circ g)(x) = \bigcup_{y \in g(x)} f(y)$ for every input x. For two function classes $\mathcal{F}$ and $\mathcal{G}$, let $\mathcal{F} \circ \mathcal{G} = \{f \circ g \mid f \in \mathcal{F}, g \in \mathcal{G}\}$. In particular, we inductively define $\mathrm{CFLSV}^{(1)} = \mathrm{CFLSV}$ and $\mathrm{CFLSV}^{(k+1)} = \mathrm{CFLSV} \circ \mathrm{CFLSV}^{(k)}$ for each index $k \geq 1$. For instance, the function $f(x) = \{xx\}$ defined for any $x \in \Sigma^*$ belongs to $\mathrm{CFLSV}^{(2)}$. This fact yields, e.g., $\mathrm{CFLSV}(2) \subseteq \mathrm{CFLSV}^{(4)}$. Unlike NP function classes (such as, NPSV and NPMV), $\mathrm{CFLSV_t}$ (and therefore CFLSV and CFLMV) is not closed under functional composition.

Proposition 5. $\mathrm{CFLSV_t} \neq \mathrm{CFLSV_t}^{(2)}$. *(Also for* CFLSV *and* CFLMV.*)*

Proof Sketch. Let $\Sigma = \{0, 1, \natural\}$ be our alphabet and define $f_{dup\natural}(x) = \{x\natural x\}$ for any input $x \in \{0,1\}^*$ and $f_{dup\natural}(x) = \{\lambda\}$ for any other inputs x. It is not difficult to show that $f_{dup\natural} \in \mathrm{CFLSV_t}^{(2)}$. To show that $f_{dup\natural} \notin \mathrm{CFLSV_t}$, we assume that $f_{dup\natural} \in \mathrm{CFLSV_t}$. Let $DUP_\natural$ be a "marked" version of DUP (duplication), defined as $DUP_\natural = \{x\natural x \mid x \in \{0,1\}^*\}$. It holds that, for every $w \neq \lambda$, $w \in \mathrm{ran}(f_{dup\natural})$ iff there is a string x such that $w \in f_{dup\natural}(x)$ iff $w \in DUP_\natural$. Thus, $DUP_\natural \cup \{\lambda\} = \mathrm{ran}(f_{dup\natural})$. Note that $DUP_\natural \in \mathrm{CFL}$ since $f_{dup\natural} \in \mathrm{CFLSV_t}$. However, it is well-known that $DUP_\natural \notin \mathrm{CFL}$. This leads to a contradiction. Therefore, we conclude that $f_{dup\natural} \notin \mathrm{CFLSV_t}$. □

To examine the role of functions in a process of recognizing a given language, a *//-advice operator*, defined by Köbler and Thierauf [9], is quite useful. Given a class $\mathcal{F}$ of functions, a language L is in $\mathrm{CFL}//\mathcal{F}$ if there exists a language $B \in \mathrm{CFL}$ and a function $h \in \mathcal{F}$ satisfying $L = \{x \mid \exists y \in h(x) \text{ s.t. } [\,{}^{x}_{y}] \in B\}$. Analogously, $\mathrm{REG}//\mathcal{F}$ is defined using REG instead of CFL. This operator naturally extends a /-advice operator of [15, 17].

In the polynomial-time setting, it holds that $\mathrm{NP} \cap \text{co-NP} = \mathrm{P}//\mathrm{NPSV_t}$ [9]. A similar equality, however, does not hold for CFL functions.

Proposition 6. *1.* $\mathrm{REG}//\mathrm{NFASV_t} \nsubseteq \mathrm{CFL}$ *and* $\mathrm{CFL} \nsubseteq \mathrm{REG}//\mathrm{NFAMV}$.
2. $\mathrm{REG}//\mathrm{NFASV_t}$ *is closed under complement but* $\mathrm{REG}//\mathrm{NFAMV}$ *is not.*
3. $\mathrm{CFL} \cap \text{co-CFL} \subsetneqq \mathrm{REG}//\mathrm{CFLSV_t}$.

Proof Sketch. (1) Note that the language $DUP_\# = \{x\#x \mid x \in \{0,1\}^*\}$ (duplication) falls into $\mathrm{REG}//\mathrm{NFASV_t}$ by setting $h(x\#y) = y$ and $B = \{[\,{}^{x}_{x}] \mid x \in \{0,1\}^*\}$. The key idea is the following claim. (*) A language L is in $\mathrm{REG}//\mathrm{NFAMV}$ iff L is recognized by a certain *one-way two-head (non-sensing) nfa* (or an nfa(2), in short) with λ-moves. See a survey, e.g., [7], for this model. Now, consider $L_{pal} = \{x\#x^R \mid x \in \{0,1\}^*\}$ (palindromes). Since L_{pal} cannot be recognized by any nfa(2), it follows that $L_{pal} \notin \mathrm{REG}//\mathrm{NFAMV}$.

(2) The non-closure property of REG//NFAMV follows from a fact that the class of languages recognized by nfa(2)'s is not closed under complement. Use the above claim (*) to obtain the desired result. □

Bwfore closing this section, we exhibit a simple structural difference between languages and functions. It is well-known that all languages over the unary alphabet $\{1\}$ in CFL belong to REG. On the contrary, there is a function $f : \{1\}^* \to \{0,1\}^*$ such that f is in CFLMV but not in NFAMV.

4 Oracle Computation and Two Relativizations

Oracle computation is a natural extension of ordinary stand-alone computation by providing external information by way of *query-and-answer* communication.

Such oracle computation can realize various forms of relativizations, including many-one and Turing relativizations and thus introduce relativized languages and functions. By analogy with relativized NP functions (e.g., [14]), let us consider many-one and Turing relativizations of CFL functions. The first notion of *many-one relativization* was discussed for languages in automata theory [8, 21] and we intend to extend it to CFL functions. Given any language A over alphabet Γ, a function $f : \Sigma^* \to \mathcal{P}(\Gamma^*)$ is in CFLMV_m^A (or $\mathrm{CFLMV}_m(A)$) if there exists an npda M with two write-only tapes (one of which is a standard output tape and the other is a *query tape*) such that, for any $x \in \Sigma^*$, (i) along any accepting computation path p of M on x (notationally, $p \in ACC_M(x)$), M produces a query string y_p on the query tape as well as an output string z_p on the output tape and (ii) $f(x)$ equals the set $\{z_p \mid y_p \in A, p \in ACC_M(x)\}$. Such an M is referred to as an *oracle npda*. Given any language family $\mathcal{C}$, we further set $\mathrm{CFLMV}_m^{\mathcal{C}}$ (or $\mathrm{CFLMV}_m(\mathcal{C})$) to be $\bigcup_{A\in\mathcal{C}} \mathrm{CFLMV}_m^A$. Similarly, we define CFLSV_m^A and $\mathrm{CFLSV}_m^{\mathcal{C}}$ by additionally demanding the size of each output string set is at most 1. Using CFLSV_m^A, a relativized language family CFL_m^A defined in [21] can be expressed as $\mathrm{CFL}_m^A = \{L \mid \eta_L \in \mathrm{CFLSV}_m^A\}$.

Lemma 7. *1.* $\mathrm{CFLMV}_m^{\mathrm{REG}} = \mathrm{CFLMV}$ *and* $\mathrm{CFLSV}_m^{\mathrm{REG}} = \mathrm{CFLSV}$.
2. $\mathrm{CFLSV}^{(k+1)} = \mathrm{CFLSV}_m^{\mathrm{CFL}(k)}$.

Proposition 8. *1.* $\mathrm{REG}//\mathrm{CFLSV_t} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(2)} \cap \text{co-}\mathrm{CFL}_m^{\mathrm{CFL}(2)}$.
2. $\mathrm{CFL}//\mathrm{CFLSV_t} \subseteq \mathrm{CFL}_m^{\mathrm{CFL}(3)}$.

Proof Sketch. We will prove only (2). For convenience, we write $[x, y]^T$ for $[\,{x \atop y}\,]$ in this proof. Let $\mathrm{CFL}_{m[1]}^A = \mathrm{CFL}_m^A$ and $\mathrm{CFL}_{m[k+1]}^A = \mathrm{CFL}_m(\mathrm{CFL}_{m[k]}^A)$ for every index $k \geq 1$ [21]. Let $L \in \mathrm{CFL}//\mathrm{CFLSV_t}$ and take an npda M and a function $h \in \mathrm{CFLSV_t}$ for which $L = \{x \mid M \text{ accepts } [x, h(x)]^T\}$. Let M' be an npda with an output tape computing h. An oracle npda N is also defined as follows. On input x, N simulates M' on x and nondeterministically produces $[\tilde{x}, \tilde{y}]^T$ on its query tape when M' outputs y, where $\tilde{x}$ and $\tilde{y}$ are appropriate $\natural$-extensions of x and y, respectively. An oracle A receives a $\natural$-extension $[\tilde{x}, \tilde{y}]^T$ and decides whether M accepts $[x, y]^T$ by removing all $\natural$s. Clearly, L belongs to CFL_m^A via N. Define another npda N_1. On input $w = [\tilde{x}, \tilde{y}]^T$, N_1 simulates M on $[x, z]^T$ by guessing z symbol by symbol. At the same time, it writes $[\tilde{y}', \tilde{z}]^T$ on a query tape and accepts w exactly when M enters an accepting state. Let $B = \{[\tilde{y}', \tilde{z}]^T \mid y = z\}$. Note that B is in $\mathrm{CFL}_m^{\mathrm{CFL}}$. Hence, A is in $\mathrm{CFL}_m^B \subseteq \mathrm{CFL}_{m[2]}^{\mathrm{CFL}}$, and thus L is in $\mathrm{CFL}_{m[3]}^{\mathrm{CFL}}$. Since $\mathrm{CFL}_{m[3]}^{\mathrm{CFL}} = \mathrm{CFL}_m^{\mathrm{CFL}(3)}$ [21], the desired conclusion follows. □

The second relativization is *Turing relativization.* A multi-valued partial function f belongs to CFLMV_T^A (or $\mathrm{CFLMV}_T(A)$) if there exists an oracle npda M having *three extra inner states* $\{q_{query}, q_{yes}, q_{no}\}$ that satisfies the following three conditions: on each input x, (i) if M enters a query state q_{query}, then a valid string, say, s written on the query tape is sent to A and, automatically, *the content of the query tape becomes blank* and *the tape head returns to the start cell,* (ii) oracle A sets M's inner state to q_{yes} if $s \in A$ and q_{no} otherwise, and (iii) all computation paths of M terminate in time $O(n)$ *no matter what oracle is used.*

Obviously, $\mathrm{CFLMV}_T^A = \mathrm{CFLMV}_T^{\overline{A}}$ holds for any oracle A. Define $\mathrm{CFLMV}_T^{\mathcal{C}}$ (or $\mathrm{CFLMV}_T(\mathcal{C})$) to be the union $\bigcup_{A\in\mathcal{C}} \mathrm{CFLMV}_T^A$ for a given language family $\mathcal{C}$.

Analogously to the well-known *NPMV hierarchy*, composed of $\Sigma_k^{\mathrm{p}}\mathrm{MV}$ and $\Pi_k^{\mathrm{p}}\mathrm{MV}$ for $k \in \mathbb{N}^+$ [14], we inductively define $\Sigma_1^{\mathrm{CFL}}\mathrm{MV} = \mathrm{CFLMV}$, $\Pi_k^{\mathrm{CFL}}\mathrm{MV} = \text{co-}\Sigma_k^{\mathrm{CFL}}\mathrm{MV}$, and $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV} = \mathrm{CFLMV}_T(\Pi_k^{\mathrm{CFL}})$ for every index $k \geq 1$. In a similar fashion, we define $\Sigma_k^{\mathrm{CFL}}\mathrm{SV}$ using CFLSV_T^A in place of CFLMV_T^A. The above *CFLMV hierarchy* is useful to scaling the computational complexity of given functions. For example, the function $f(w) = \{x \in \{0,1\}^* \mid \exists u, v\,[w = uxxv]\}$ for every $w \in \{0,1\}^*$ belongs to $\Sigma_2^{\mathrm{CFL}}\mathrm{MV}$. Moreover, it is possible to show that $\mathrm{CFLMV} \cup \text{co-CFLMV} \subseteq \mathrm{CFLMV} \ominus \mathrm{CFLMV} \subseteq \Sigma_4^{\mathrm{CFL}}\mathrm{MV}$.

Proposition 9. *Each function in $\bigcup_{k\in\mathbb{N}^+} \Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$ can be computed by an appropriate $O(n)$ space-bounded multi-tape deterministic Turing machine.*

Proof Sketch. It is known in [21] that $\Sigma_k^{\mathrm{CFL}} \subseteq \mathrm{DSPACE}(O(n))$ for every $k \geq 1$. It therefore suffices to show that, for any fixed language $A \in \mathrm{DSPACE}(O(n))$, every function f in ${\mathrm{CFLSV}_{\mathrm{t}}}^A$ can be computed using $O(n)$ space. This is done by a direct simulation of f on a multi-tape Turing machine. □

For $k \geq 3$, it is possible to give the exact characterization of $\mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. This makes a sharp contrast with Proposition 6(3).

Proposition 10. *For every index $k \geq 3$, $\Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}} = \mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$.*

Proof Sketch. By extending the proof of Proposition 6(3), it is possible to show that $\Sigma_k^{\mathrm{CFL}} \cap \Pi_k^{\mathrm{CFL}} \subseteq \mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$. Similarly to Proposition 6(2), it holds that $\mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}}$ is closed under complement. It thus suffices to show that $\mathrm{REG}/\!/\Sigma_k^{\mathrm{CFL}}\mathrm{SV}_{\mathrm{t}} \subseteq \Sigma_k^{\mathrm{CFL}}$. This can be done by a direct simulation. □

Lemma 11. *Let $e \geq 2$ and $k \geq 1$. $\Sigma_k^{\mathrm{CFL}}\mathrm{SV} = \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV} \Rightarrow \Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}} \Rightarrow \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV} = \Sigma_{k+e}^{\mathrm{CFL}}\mathrm{SV}$.*

Proof Sketch. Assuming $\Sigma_k^{\mathrm{CFL}}\mathrm{SV} = \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$, let us take any language $A \in \Sigma_{k+1}^{\mathrm{CFL}}$ and consider η_A. It is not difficult to show that, for any index $d \in \mathbb{N}^+$, $A \in \Sigma_d^{\mathrm{CFL}}$ iff $\eta_A \in \Sigma_d^{\mathrm{CFL}}\mathrm{SV}$. Thus, $\eta_A \in \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV} = \Sigma_k^{\mathrm{CFL}}\mathrm{SV}$. This implies that $A \in \Sigma_k^{\mathrm{CFL}}$. Next, assume that $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$. It is proven in [21] that $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$ iff $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+e}^{\mathrm{CFL}}$ for all $e \geq 1$. Hence, for every $e \geq 2$, we obtain $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+e-1}^{\mathrm{CFL}}$, which is equivalent to $\Pi_k^{\mathrm{CFL}} = \Pi_{k+e-1}^{\mathrm{CFL}}$. It then follows that $\Sigma_{k+e}^{\mathrm{CFL}}\mathrm{SV} = \mathrm{CFLSV}_T(\Pi_{k+e-1}^{\mathrm{CFL}}) = \mathrm{CFLSV}_T(\Pi_k^{\mathrm{CFL}}) = \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$. □

Regarding refinement, from the proof of [8, Theorem 3] follows $\mathrm{NFAMV} \sqsubseteq_{ref} \mathrm{NFASV}$. This result leads to $\mathrm{NFAMV} \circ \mathrm{CFLSV} \sqsubseteq_{ref} \mathrm{CFLSV}$ in [10]. By a direct simulation, nevertheless, it is possible to show that $\Sigma_k^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$ for every $k \geq 1$. Lemma 11 together with this fact leads to the following consequence.

Proposition 12. *Let $k \geq 2$. If $\Sigma_k^{\mathrm{CFL}} = \Sigma_{k+1}^{\mathrm{CFL}}$, then $\Sigma_{k+1}^{\mathrm{CFL}}\mathrm{MV} \sqsubseteq_{ref} \Sigma_{k+1}^{\mathrm{CFL}}\mathrm{SV}$.*

Recently, it was shown in [22] that $\mathrm{CFLMV} \not\sqsubseteq_{ref} \mathrm{CFLSV}$ holds. However, it is not known if this can be extended to every level of the CFLMV hierarchy.

5 Optimization Functions

An *optimization problem* is to find an optimal feasible solution that satisfy a given condition. Krentel [11] studied the complexity of those optimization problems. Analogously to OptP of Krentel, we define OptCFL as a collection of *single-valued total functions* $f : \Sigma^* \to \Gamma^*$ such that there exists an npda M and an *opt* $\in$ {maximum,minimum} for which, for every string $x \in \Sigma^*$, $f(x)$ denotes the *opt* output string of M on input x along an appropriate accepting computation path, assuming that M must have at least one accepting computation path. Here, we use the *dictionary (or alphabetical) order* $<$ over Γ^* (e.g., *abbe* $<$ *abc* and *ab* $<$ *aba*) instead of the lexicographic order to compensate the npda's limited ability of comparing two strings *from left to right.* For example, the function $f(w) = \max\{PAL_{sub}(w)\}$ for $w \in \{0,1\}^*$, where PAL_{sub} is defined in Section 1, is a member of OptCFL. It holds that $\mathrm{CFLMV_t} \sqsubseteq_{ref}$ OptCFL.

Proposition 13. $\mathrm{CFLSV_t} \subsetneqq \mathrm{OptCFL} \subseteq \Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$.

The first part of Proposition 13 comes from a fact that the function $f(w) = \max\{g(w)\}$, where $g(w) = \{\lambda\} \cup \{x_i y_i \mid w = x_1\natural x_2\natural x_3\#y_1\natural y_2\natural y_3, x_i = y_i^R, i \in \{1,2,3\}\}$, is in OptCFL but not in CFLSV_t. This latter part is proven by applying the functional pumping lemma.

Note that, in the polynomial-time setting, a much sharper upper-bound of $\mathrm{OptP} \subseteq \Sigma_2^{\mathrm{p}}\mathrm{SV_t}$ is known. Similarly to OptCFL, let us define OptNFA_{EL} using nfa's M instead of npda's with an extra condition that $M(x)$ outputs only strings of the *equal length.* This new class is located within $\Sigma_2^{\mathrm{CFL}}\mathrm{SV_t}$.

Proposition 14. $\mathrm{OptNFA}_{EL} \subseteq \Sigma_2^{\mathrm{CFL}}\mathrm{SV_t}$.

Proof Sketch. Let $f \in \mathrm{OptNFA}_{EL}$ and take an underlying nfa N that forces $f(x)$ to equal $\max\{N(x)\}$ for every x. Define an oracle npda M_1 to simulate N on x and output, say, y. Simultaneously, query $y\#x^R$ using a stack wisely. If its oracle answer is 1, enter an accepting state; otherwise, reject. Make another npda M_2 receive $y\#x^R$, simulate N^R on x^R, and compare its outcome with y^R, where the notation N^R refers to an nfa that reverses the computation of N. □

Proposition 15. *1.* $\mathrm{CFL} \cup \text{co-CFL} \subseteq \mathrm{REG//OptCFL} \subseteq \Sigma_4^{\mathrm{CFL}} \cap \Pi_4^{\mathrm{CFL}}$.
2. $\mathrm{CFL//OptCFL} \subseteq \Sigma_5^{\mathrm{CFL}}$.

Proof Sketch. We will show only (1). Note that REG//OptCFL is closed under complementation. Let $L \in$ CFL and take an npda M recognizing L. Define N_1 as follows. On input x, guess a bit b. If $b = 0$, then output 0 in an accepting state. Otherwise, simulate M on x and output 1 along only accepting computation paths of M. Let $h(x)$ be $\max\{N_1(x)\}$ for all x's. It follows that $L = \{[\genfrac{}{}{0pt}{}{x}{h(x)}] \mid h(x) \neq \emptyset\}$. This proves that $L \in$ REG//OptCFL. Next, let $L \in$ REG//OptCFL. Since $\mathrm{OptCFL} \subseteq \Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$, we obtain $L \in \mathrm{REG//}\Sigma_4^{\mathrm{CFL}}\mathrm{SV_t}$. By Proposition 10, this implies that L belongs to $\Sigma_4^{\mathrm{CFL}} \cap \Pi_4^{\mathrm{CFL}}$. □

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: A general theory of translation. Math. Systems Theory, 3, 193–221 (1967)
2. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase-structure grammars. Z. Phonetik Sprachwiss. Kommunik., 14, 143–172 (1961)
3. Choffrut, C., Culik, K.: Properties of finite and pushdown transducers. SIAM J. Comput., 12, 300–315 (1983)
4. Evey, R.J.: Application of pushdown-store machines. In: Proc. 1963 Fall Joint Computer Conference, AFIPS Press, pp.215–227, 1963
5. Fenner, S.A., Homer, S., Ogihara, M., Selman, A.L.: Oracles that compute values. SIAM J. Comput., 26, 1043–1065 (1997)
6. Fisher, P.C.: On computability by certain classes of restricted Turing machines. In: Proc. 4th Annual IEEE Symp. on Switching Circuit Theory and Logical Design (SWCT'63), IEEE Computer Society, pp.23–32, 1963
7. Holzer, M., Kutrib, M., Malcher, A.: Multi-head finite automata: characterizations, concepts and open problems. In: Proc. of the Workshop on The Complexity of Simple Programs, number 1 in EPTCS, pp.93–107, 2008
8. Kobayashi, K.: Classification of formal langauges by functional binary transductions. Inform. Control, 15, 95–109 (1969)
9. Köbler, J., Thierauf, T.: Complexity-restricted advice functions. SIAM J. Comput., 23, 261–275 (1994)
10. Konstantinidis, S., Santean, N., Yu, S.: Representation and uniformization of algebraic transductions. Acta Inform., 43, 395–417 (2007)
11. Krentel, M.: The complexity of optimization problems. J. Comput. System Sci., 36, 490–509 (1988)
12. Liu, L.Y., Weiner, P.: An infinite hierarchy of intersections of context-free languages. Math. Systems Theory, 7, 185–192 (1973)
13. Selman, A.L.: A taxonomy of complexity classes of functions. J. Comput. System Sci., 48, 357–381 (1994)
14. Selman, A.L.: Much ado about functions. In: Proc. of the 11th Annual IEEE Conference on Computational Complexity, pp.198–212, 1996
15. Tadaki, K., Yamakami, T., Lin, J. C. H.: Theory of one-tape linear-time Turing machines. Theor. Comput. Sci., 411, 22–43 (2010)
16. Wotschke, D.: Nondeterminism and Boolean operations in pda's. J. Comp. System Sci., 16, 456–461 (1978)
17. Yamakami, T.: Swapping lemmas for regular and context-free languages. Available at arXiv:0808.4122 (2008)
18. Yamakami, T.: The roles of advice to one-tape linear-time Turing machines and finite automata. Int. J. Found. Comput. Sci., 21, 941–962 (2010)
19. Yamakami, T.: Immunity and pseudorandomness of context-free languages. Theor. Comput. Sci., 412, 6432–6450 (2011)
20. Yamakami, T.: Pseudorandom generators against advised context-free languages. See arXiv:0902.2774 (2009)
21. Yamakami, T.: Oracle pushdown automata, nondeterministic reducibilities, and the hierarchy over the family of context-free languages. In: Proc. of SOFSEM 2014, V. Geffert et al. (eds.), LNCS, vol. 8327, pp. 514–525 (2014). A complete version appears at arXiv:1303.1717.
22. Yamakami, T.: Not all multi-valued partial CFL functions are refined by single-valued functions. In: Proc. of IFIP TCS 2014, J. Diaz et al. (eds), LNCS vol. 8705, pp. 136–150 (2014)

Proving termination of programs having transition invariants of height ω

Stefano Berardi[1], Paulo Oliva[2], and Silvia Steila[1]

[1] Università degli studi di Torino
[2] Queen Mary University of London

Abstract. We study the proof of a recent and relevant result about termination of programs, the Termination Theorem by Podelski and Rybalchenko [9]. We prove that in a special case, the only case which is used in applications, all programs proved to be terminating may be described by some primitive recursive map.

1 Introducing the Termination Theorem

Fix any transition relation R over the set S of possible states of a program P. Assume $\mathtt{In} \subseteq S$ is the set of possible initial states of P, and that $\mathtt{Acc}$ is the set of accessible states of P, if we start from some state in $\mathtt{In}$ and we use the relation R finitely many times. The Termination Theorem by Podelski and Rybalchenko [9] may be stated as follows. The transition relation R is terminating from any initial state if and only if the transitive closure R^+ of R, restricted to the set $\mathtt{Acc}$ of accessible states, is included in some finite union of well-founded relations.

The authors formulate the Termination Theorem by introducing the concept of "disjunctively well-founded transition invariant". A disjunctively well-founded transition invariant is any binary relation T which is the union of a family $T_1, \ldots, T_n$ of well-founded relations, and which includes the restriction to $\mathtt{Acc}$ of R^+. The original statement of the Termination Theorem is: "R is terminating from any initial state if and only if R has some disjunctively well-founded transition invariant T".

By building over this result the same authors and Byron Cook designed an algorithm they called Terminator [5], checking a sufficient condition for termination for a while-if program P in a simplified programming language. The Terminator algorithm takes P and looks for a disjunctively well-founded transition invariant $T = T_1 \cup \ldots \cup T_n$ for P, with $T_1, \ldots, T_n$ well-founded relations *of height* ω. The extra feature "of height ω" is found in the algorithm but not in the Theorem. If the Terminator algorithm finds $T_1, \ldots, T_n$ as above, it deduces the termination for the program P using the Termination Theorem.

This particular application of the Termination Theorem raises an interesting question: what is the status of a transition relation R having a disjunctively well-founded transition invariant $T = T_1 \cup \ldots \cup T_n$where each T_i has height ω? An answer to this question can lead to a characterization of the set of while-if programs which the termination algorithm can prove to be terminating.

2 A characterization of the Termination Theorem in the case of invariants of height ω

Our first result is the following. The Termination Theorem may derive that a transition relation R is terminating using n relations $T_1, \ldots, T_n$ of height ω if and only if R has height $\leq \omega^n$. Besides, in the case $T_1, \ldots, T_n$ are primitive recursive and R itself is (the graph of) the restriction of some primitive recursive map to some primitive recursive subset, we may say more. In this case, indeed, the final state of the program P is computable by some primitive recursive map in the initial state.

As a corollary we derive that the set of functions, having at least one implementation in Podelski-Rybalchenko while-if language with a well-founded disjunctively transition invariant where each relation has height ω, is exactly the set of primitive recursive functions. This is an ongoing work: a preliminary draft may be found in [1]. An independent proof of the same result, again in the form of preliminary draft, may be found in [8]. The authors follow a completely different approach, they use a miniaturization of the Dickson Lemma to prove the Termination Theorem.

3 A sketch of our proof

Our approach is based over the analysis a new intuitionistic proof of the Termination Theorem [2] (another intuitionistic proof already existed, by Thierry Coquand [6]). The original proof of the Termination Theorem requires classical logic and Ramsey's Theorem. In order to intuitionistically prove the Termination Theorem we introduced a kind of contrapositive of Ramsey Theorem, the H-closure Theorem [2], which we are going to explain.

First of all, we introduce the notion of H-well-foundation. Let T be any binary relation on some set I. We say that a sequence s is T-homogeneous if $s \in H(T)$, where $H(T)$ is defined as follows.

Let T be a binary relation on some set I. $H(T)$ is the set of the T-decreasing transitive finite sequences on I:

$$\langle x_1, \ldots, x_n \rangle \in H(T) \iff \forall i, j \in [1, n]. i < j \implies x_j T x_i.$$

T is H-well-founded if $H(T)$ is well-founded by one-step extension. If T is well-founded that T is H-well-founded, but H-well-foundation is much weaker than well-foundation. The notion of H-closure is new, therefore we provide some examples. The relation $T \equiv (\neq)$ over $\{0, 1\}$ is not well-founded because we have the infinite chain $0 \neq 1 \neq 0 \neq 1 \ldots$. Any sequence $s \in H(T)$, by definition unfolding, has any two elements in relation $\neq$, therefore has pairwise distinct elements, hence has length ≤ 2. Thus, $H(T)$ has height 2 w.r.t. the one-step extension relation, therefore $H(T)$ is well-founded, and T is H-well-founded. Another example (for which we skip the proof): a relation T over a finite set is well-founded if and only if there are no T-cycles, that is, there are no $x_0, \ldots, x_n \in$

I such that $x_0 T x_1 T \dots T x_n = x_0$. A relation T over a finite set is H-well-founded if and only if there are no T-loops, that is, there is no $x \in I$ such that xTx. This second condition is much weaker that the first one, a loop is a cycle but a cycle in general is not a loop.

The H-closure Theorem says that if $R_1, \dots, R_k$ are H-well-founded then $(R_1 \cup \dots \cup R_k)$ is also H-well-founded. H-closure has an intuitionistic proof, and, as we said, intuitionistically derives the Termination Theorem. In order to characterize the Termination Theorem in the case of height ω relations, we first strengthen H-closure as follows. If each R_i has ordinal height less or equal than α_i, then $H(R_1 \cup \dots \cup R_k)$ has ordinal height less or equal than $2^{\alpha_1 \oplus \dots \oplus \alpha_k}$, where $\oplus$ is the natural sum of ordinals, defined as the smallest binary function which is increasing in both arguments w.r.t. the pointwise ordering [4]. The proof uses a simulation of the ordering of $H(R_1 \cup \dots \cup R_k)$ in the inclusion ordering over the set of k-branching trees, whose branches are decreasing sequences in $R_1 \oplus \dots \oplus R_k$ [1].

Eventually, we embed the ordering of $H(R_1 \cup \dots \cup R_k)$ into the ordering over $[0, \omega^k]$, and we use the characterization for the decreasing sequences over $[0, \omega^k]$ in order to characterize the sequences of transitions for a given program P.

After this proof was done, we were informed that Delhommé [7] and Blass and Gurevich [3] have already observed that the computation of the ordinal height of a relation proven to be well-founded by the Termination Theorem is the natural product of the individual heights.

4 Conclusion and future work

We proved the following characterization of the Termination Theorem. Assume we have a program P whose transition relation R is the graph of a partial recursive map restricted to a primitive recursive domain. Assume we have a disjunctively well-founded transition invariant $T = T_1 \cup \dots\dots T_n$ for R, with $T_1, \dots, T_n$ primitive recursive and of height ω. Then we may compute the number of steps of R and the final state by some primitive recursive function in the initial state.

We conjecture that the same result holds for the Terminator Algorithm based on the Termination Theorem: a function has at least one implementation in Podelski-Rybalchenko language which the Terminator Algorithm may catch terminating if and only if the function is primitive recursive. One of the authors is working on a proof of it. The result is not self-evident because there is much more in the Terminator algorithm than just the Termination Theorem.

If compared to the characterization of Termination Theorem based on Dickson Lemma, our characterization has the advantage of being based over the original proof of the Theorem. For this reason, we hope in a future work to be able characterize the Termination Theorem in general, in the case of well-founded relations of any ordinal height.

References

1. S. Berardi, P. Oliva, and S. Steila. Proving termination with transition invariants of height omega. Preliminary Draft, 2014.
2. S. Berardi and S. Steila. Ramsey theorem as an intuitionistic property of well founded relations. pages 93–107, 2014. RTA-TLCA.
3. A. Blass and Y. Gurevich. Program termination and well partial orderings. *ACM Trans. Comput. Logic*, 9(3):18:1–18:26, 2008.
4. P. Carruth. Arithmetic of ordinals with applications to the theory of ordered abelian groups. *Bull. Amer. Math. Soc.*, 48(4):223–334.
5. B. Cook, A. Podelski, and A. Rybalchenko. Abstraction refinement for termination. In *SAS*, pages 87–101, 2005.
6. T. Coquand. An analysis of ramsey's theorem. *Inf. Comput.*, 110(2):297–304, 1994.
7. C. Delhommé. Height of a Superposition. *Order*, 23(2-3):221–233, 2006.
8. D. Figueira, S. Figueira, S. Schmitz, and Ph. Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's Lemma. In *LICS 2011: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science*, pages 269–278. IEEE Press, 2011.
9. A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, pages 32–41. IEEE Computer Society, 2004.

Orthomodular algebraic lattices related to combinatorial posets

Extended Abstract

Luca Bernardinello, Lucia Pomello, and Stefania Rombolà

DISCo, Università degli studi di Milano–Bicocca
viale Sarca 336 U14, Milano, Italia

1 Introduction

We extend some theoretical results in the frame of concurrency theory, which were presented in [1]. In particular, we focus on partially ordered sets (posets) as models of nonsequential processes [2] and we apply the same construction as in [1] of a lattice of subsets of points of the poset via a closure operator defined on the basis of the concurrency relation, viewed as lack of causal dependence.

The inspiring idea is related to works by C. A. Petri [6]. Petri proposed a theory of systems based on abstract models to represent the behaviour and the properties of concurrent and distributed systems, which takes into account the principles of the special relativity. A crucial difference between the standard physical theories and the framework in which Petri develops his own theory comes from the use of the continuum as the underlying model in physics.

In the combinatorial model proposed by Petri, the usual notion of density of the continuum model is replaced by two properties strictly related and required for the posets modelling a discrete space-time: the so-called K-density and a weaker form called N-density. K-density is based on the idea that any maximal antichain (or *cut*) in a poset and any maximal chain (or *line*) have a non-empty intersection. A line can be interpreted as a sequential subprocess, while a cut corresponds to a time instant and K-density requires that, at any time instant, any sequential subprocess must be in some state or changing its state. N-density can be viewed as a sort of local density and was introduced by Petri as an axiom for posets modelling nonsequential processes. Occurrence nets, a fundamental model of such processes, are indeed N-dense, whereas for example event structures [5] are in general not N-dense.

In [1] we have considered as model of non sequential processes a class of locally finite posets and shown that the closed subsets, obtained via a closure operator defined on the basis of concurrency, correspond in general to subprocesses which result to be ‘closed’ with respect to the Petri net firing rule. Moreover, we have shown that if the poset is N-dense, then the lattice of closed subsets is *orthomodular*. Orthomodular lattices are families of partially overlapping Boolean algebras and have been studied as the algebraic model of quantum logic [7].

In this paper we generalize our previous results for combinatorial posets and we show that the N-density of the poset is a sufficient and necessary condition for the orthomodularity of the lattice of closed subsets.

In an orthomodular lattice, each element is associated to its *orthocomplement*. We show that under K-density, given a closed set, any line intersects the closed set or its orthocomplement. Starting from this result we define the characteristic map of the family of closed sets that cross the given line and show that this map is a two-valued state of the lattice of closed sets of the poset. The notion of two-valued state over orthomodular lattices is used in quantum logic, where the elements of the lattice are interpreted as propositions of a language, and the two-valued states on a lattice as consistent assignments of truth values to these propositions. This suggests to look at the closed sets as propositions in a logic language, where orthocomplementation corresponds to negation and any line induces a logical interpretation. Following the same idea, we propose to consider the dual relation between cuts and Boolean algebras, on the fact that any cut of a K-dense poset generates a Boolean subalgebra of the lattice.

Finally, we extend to combinatorial posets the relation between the K-density of a poset and the algebraicity of the lattice of its closed sets, as given in [1].

2 Preliminary Definitions

In this section, we recall basic definitions and notations for partially ordered sets, lattices and orthomodular lattices, and closure operators.

A *partially ordered set* (poset for short) is a set P, together with a reflexive, anti-symmetric, and transitive relation $\leq \subseteq P \times P$. By $<$ we denote the related strict partial order.

For $x, y \in P$, we write $x \lessdot y$ if $x < y$ and no $z \in P$ satisfies $x < z < y$. Let ${}^\bullet x = \{\, y \mid y \lessdot x \,\}$, and $x^\bullet = \{\, y \mid x \lessdot y \,\}$. A poset $\mathcal{P} = (P, \leq)$ is *combinatorial* iff $\leq\, = (\lessdot)^*$, where $(\lessdot)^*$ is the transitive and reflexive closure of $\lessdot$. A poset $\mathcal{P} = (P, \leq)$ is of *finite degree* iff $\forall x \in P : |{}^\bullet x| \in \mathbb{N}$ and $|x^\bullet| \in \mathbb{N}$.

Given a partial order relation $\leq$ on a set P, we can derive the relations $li = \leq \cup \geq$, and $co = (P \times P) \setminus li$. Intuitively, in our framework $x\ li\ y$ means that x and y are connected by a causal relation, and $x\ co\ y$ means that x and y are causally independent. The relations li and co are symmetric but, in general, non transitive. Note that li is a reflexive relation, while co is irreflexive.

A *clique* of a binary relation is a set of pairwise related elements; a clique of $co \cup id_P$ will be called *antichain*, or *co-set*, whereas a clique of li will be called *chain* or *li-set*. Maximal cliques of $co \cup id_P$ and li are called, respectively, *cuts* and *lines*: $cuts(\mathcal{P}) = \{c \subseteq P \mid c \text{ is a maximal clique of } co \cup id_P\}$, $lines(\mathcal{P}) = \{\, \lambda \subseteq P \mid \lambda \text{ is a maximal clique of } li\}$

Definition 1. $\mathcal{P} = (P, \leq)$ *is* K-dense $\Leftrightarrow \forall c \in \text{cuts}(\mathcal{P}), \forall \lambda \in \text{lines}(\mathcal{P}) : c \cap \lambda \neq \emptyset$.

Obviously, in general $|c \cap \lambda| \leq 1$.

In the following we are interested in a weaker form of density, called *N-density*, strictly related to K-density and which can be viewed as a sort of local density.

Definition 2. $\mathcal{P} = (P, \leq)$ *is* N-dense $\Leftrightarrow \forall\, x, y, v, w \in P$*:* $(y < v$ *and* $y < x$ *and* $w < v$ *and* $(y$ *co* w *co* x *co* $v)) \Rightarrow \exists z \in P : (y < z < v$ *and* $(w$ *co* z *co* $x))$*.*

Definition 3. *An* orthocomplemented poset $\mathcal{P} = \langle P, \leq, 0, 1, (.)' \rangle$ *is a partially ordered set* $\mathcal{P} = (P, \leq)$*, equipped with a minimum and a maximum element, respectively denoted by 0 and 1, and with a map* $(.)' : P \to P$*, such that the following conditions are satisfied (where* $\vee$ *and* $\wedge$ *denote, respectively, the least upper bound and the greatest lower bound with respect to* $\leq$*, when they exist):* $\forall x, y \in P$*,* (i) $(x')' = x$*,* (ii) $x \leq y \Rightarrow y' \leq x'$*,* (iii) $x \wedge x' = 0$ and $x \vee x' = 1$.

The map $(.)' : P \to P$ is called an *orthocomplementation* in $\mathcal{P}$. In an orthocomplemented poset, $\wedge$ and $\vee$, when they exist, are not independent: in fact, the so-called De Morgan laws hold: $(x \vee y)' = x' \wedge y'$, $(x \wedge y)' = x' \vee y'$. In the following, we will sometimes use *meet* and *join* to denote, respectively, $\wedge$ and $\vee$.

Two elements $x, y \in P$ are *orthogonal*, denoted $x \perp y$, iff $x \leq y'$.

Definition 4. *A* two-valued state *on a poset* $\mathcal{P}$ *is a mapping* $s : P \mapsto \{0, 1\}$ *such that (i)* $s(1) = 1$*, (ii) if* $\{a_i : i \in \mathbb{N}\}$ *is a sequence of mutually orthogonal elements in* $\mathcal{P}$*, then* $s(\bigvee_{i\in\mathbb{N}} a_i) = \sum_{i\in\mathbb{N}} s(a_i)$*.*

A poset $\mathcal{P}$ is called *orthocomplete* when it is orthocomplemented and every pairwise orthogonal countable subset of P has a least upper bound.

A lattice $\mathcal{L} = (L, \leq)$ is a poset in which, for any pair of elements, meet and join exist. A lattice $\mathcal{L}$ is *complete* when the meet and the join of any subset of $\mathcal{L}$ exist.

Definition 5. *An* orthomodular poset $\mathcal{P} = \langle P, \leq, 0, 1, (.)' \rangle$ *is an orthocomplete poset which satisfies the condition:* $x \leq y \Rightarrow y = x \vee (y \wedge x')$*, which is usually referred to as the orthomodular law.*

Let X be a set, and $\alpha \subseteq X \times X$ be a symmetric relation. Given $A \subseteq X$ we can define an operator $(.)^{\perp}$ on the powerset of X: $A^{\perp} = \{x \in X \mid \forall y \in A : (x, y) \in \alpha\}$. By applying twice the operator $(.)^{\perp}$, we get a new operator $C(.) = (.)^{\perp\perp} = ((.)^{\perp})^{\perp}$. The map C on the powerset of X is a *closure operator* on X; i.e.: for all $A, B \subseteq X$, *(i)* $A \subseteq C(A)$; *(ii)* $A \subseteq B \Rightarrow C(A) \subseteq C(B)$; *(iii)* $C(C(A)) = C(A)$ [4]. A subset A of X is *closed* if $A = A^{\perp\perp}$. The family $L(X)$ of all closed sets of X, ordered by set inclusion, is then a complete lattice [3].

When α is also irreflexive, the operator $(.)^{\perp}$, applied to elements of $L(X)$, is an orthocomplementation; the structure $\mathcal{L}(X) = \langle L(X), \subseteq, \emptyset, X, (.)^{\perp} \rangle$ then forms an orthocomplemented complete lattice [3].

A complete lattice $\mathcal{L}$ is *algebraic* if, for each $a \in L$, $a = \bigvee\{k \in K(\mathcal{L}) : k \leq a\}$, where $K(\mathcal{L})$ is the set of compact elements, and $k \in L$ is *compact* if, for every subset S in L, $k \leq \bigvee S \Rightarrow k \leq \bigvee T$, for some finite subset T of S, (see [4]).

3 A Closure Operator Based on Concurrency

In this section we consider the closed sets induced by the concurrency relation in partially ordered sets by applying the construction recalled in the previous section. We study the resulting properties of closed sets, investigating in particular the relations with N-density and K-density of the poset.

Let $\mathcal{P} = (P, \leq)$ be a poset. We can define an operator on subsets of P, which corresponds to an orthocomplementation, since *co* is irreflexive, and by this operator we define closed sets.

Definition 6. *Let $S \subseteq P$, then*

(i) $S^{\perp} = \{x \in P \mid \forall y \in S : x \; co \; y\}$ is the *orthocomplement* of S;

(ii) if $S = (S^{\perp})^{\perp}$, then S is a *closed set* of $\mathcal{P} = (P, \leq)$.

The set $S^{\perp}$ contains the elements of P which are not in causal relation with any element of S. Obviously, $S \cap S^{\perp} = \emptyset$ for any $S \subseteq P$, however in general $S \cup S^{\perp} \neq P$. In the following, we sometimes denote $(S^{\perp})^{\perp}$ by $S^{\perp\perp}$. Note that: $\forall c \in cuts(\mathcal{P})$, $c^{\perp} = \emptyset$ and $c^{\perp\perp} = P$.

We call $L(P)$ the collection of closed sets of $\mathcal{P} = (P, \leq)$. By the results on closure operators recalled in the previous section and since the relation *co* is irreflexive, we know that $\mathcal{L}(P) = \langle L(P), \subseteq, \emptyset, P, (.)^{\perp} \rangle$ is an orthocomplemented complete lattice, in which the meet is just set intersection, while the join of a family of elements is given by set union followed by closure.

Now we present our principal result for combinatorial posets: N-density is necessary and sufficient for the orthomodularity of $\mathcal{L}(P)$.

Theorem 1. *If $\mathcal{P} = (P, \leq)$ is combinatorial, then $\mathcal{L}(P)$ is orthomodular if and only if $\mathcal{P} = (P, \leq)$ is N-dense.*

Note that the orthomodular law requires that, if an element is strictly bigger than another one, then the meet between the first element and the orthocomplement of the second one should be different from the minimum element. Hence, if $A \subset B$ then B contains at least an element x concurrent with A.

The orthomodular law is weaker than the distributive law. Orthocomplemented distributive lattices are called Boolean algebras. Orthomodular lattices can therefore be considered as a generalization of Boolean algebras.

Now we characterize the K-density of a poset $\mathcal{P} = (P, \leq)$ by a property of the closed sets. In particular, we show that the combinatorial N-dense posets are K-dense if and only if, given a closed set, any line intersects either the closed set or its orthocomplement.

Theorem 2. *If $\mathcal{P} = (P, \leq)$ is combinatorial and N-dense, then*

$$\mathcal{P} = (P, \leq) \text{ is K-dense} \Leftrightarrow \forall S \in L(P), \forall \lambda \in lines(\mathcal{P}),\ \lambda \cap (S \cup S^{\perp}) \neq \emptyset$$

From Theorem 2 it follows a crucial relation between lines and closed sets; namely, given a closed set S, a line λ crosses either S or $S^{\perp}$ ($\lambda \cap S \neq \emptyset \iff \lambda \cap S^{\perp} = \emptyset$).

We now define a map associated to a line of $\mathcal{P} = (P, \leq)$: the characteristic map of the family of closed sets that cross the given line.

Definition 7. *Let $\lambda \in lines(\mathcal{P})$. Define $\Delta(\lambda) = \{S \in L(P) \mid S \cap \lambda \neq \emptyset\}$, and $\delta_{\lambda} : L(P) \to \{0, 1\}$ such that: for each $S \in L(P)$, $\delta_{\lambda}(S) = 1$ if $S \in \Delta(\lambda)$, $\delta_{\lambda}(S) = 0$ otherwise.*

Theorem 3. *Let $\mathcal{P} = (P, \leq)$ be a K-dense poset. The map δ_λ is a two-valued state of the lattice $\mathcal{L}(P)$.*

This result allows to state that any line in a combinatorial and K-dense poset $\mathcal{P} = (P, \leq)$ identifies a two-valued state in the lattice of closed sets $\mathcal{L}(P)$.

There is a dual relation between the cuts of $\mathcal{P} = (P, \leq)$ and the Boolean subalgebras in the lattice of closed sets $\mathcal{L}(P)$: any cut $\tau \in cuts(\mathcal{P})$ generates a Boolean subalgebra of $\mathcal{L}(P)$ [7].

The next theorem states that, for combinatorial posets, K-density and degree finiteness are sufficient for the algebraicity of the lattice of closed sets.

Theorem 4. *The family $L(P)$ of the closed sets of a combinatorial, K-dense and degree finite poset forms an algebraic lattice.*

In conclusion, we have proved that for combinatorial posets, N-density implies the orthomodularity of the lattice of closed sets defined on the basis of concurrency. An orthomodular lattice is always regular ([7]) and hence can be seen as a family of partially overlapping Boolean algebras.

Moreover, we have shown that for combinatorial posets the K-density determines a crucial relation between lines and closed sets: given a closed set a line crosses either the closed set or its orthocomplement. This suggests to look at the family of closed sets as the set of propositions in a logic language and at the lines as two-valued states and hence as interpretations (models) of the propositions. In general, the lattice of closed sets is not a Boolean algebra, so that the resulting logic is non-classical; we point to the cuts of the combinatorial poset as the Boolean substructures of the overall lattice.

Finally, we proved that K-density, together with degree finiteness, is a sufficient condition for the algebraicity of the lattice.

Acknowledgement
This work was partially supported by MIUR and by MIUR-PRIN 2010/2011 grant code H41J12000190001.

References

1. L. Bernardinello, L. Pomello, and S. Rombolà. Closure operators and lattices derived from concurrency in posets and occurrence nets. *Fundam. Inform.*, 105(3):211–235, 2010.
2. E. Best and C. Fernandez. *Nonsequential Processes–A Petri Net View*, volume 13 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.
3. G. Birkhoff. *Lattice Theory.* American Mathematical Society; 3rd Ed., 1979.
4. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
5. M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, part i. *Theor. Comput. Sci.*, 13:85–108, 1981.
6. C. A. Petri. State-transition structures in physics and in computation. *International Journal of Theoretical Physics*, 21(12):979–992, 1982.
7. P. Pták, P. Pulmannová. *Orthomodular Structures as Quantum Logics.* Kluwer Academic Publishers, 1991.

Abstract argumentation frameworks to promote fairness and rationality in multi-experts multi-criteria decision making

Stefano Bistarelli[1], Martine Ceberio[2], Joel A. Henderson[2], and Francesco Santini[3]

[1] Dipartimento di Matematica e Informatica
Università di Perugia
Via Vanvitelli, 1 – 06123 Perugia, Italy
bista@dmi.unipg.it

[2] Computer Science Department
The University of Texas at El Paso
500 West University – El Paso, TX 79968, USA
mceberio@utep.edu, jahenderson@miners.utep.edu

[3] Istituto di Informatica e Telematica, CNR-Pisa
Via Moruzzi, 1 – 56124, Pisa, Italy
francesco.santini@iit.cnr.it

Abstract. In this work, we propose to model Multi-Experts Multi-Criteria Decision-Making (MEMCDM) problems using Abstract Argumentation Frameworks. We specifically design our model so as to emulate fairness and rationality in the decision-making process. For instance, when, of two expert's decisions, one is unfair, we impose an attack between these two decisions, forcing one of the two decisions out of the argumentation network's resulting extensions. Similarly, we specifically put irrational decisions in opposition to force one out. In doing so, we aim to enable the prediction of decisions that are themselves fair and rational. Our model is illustrated on a toy example.

Keywords: Multi-Experts Multi-Criteria Decision Making, Disagreement, Fairness, Rationality, Argumentation Framework, Model.

1 Introduction

Expert analysis and decisions arguably provide high-quality and highly-valued support for action and policy making in a wide variety of fields, from social services, to medicine, to engineering, to grant funding committees, and so on. However, the use of experts can be prohibitive due to either lack of availability, high cost, or limited time frame for action – this is the case particularly more so in impoverished areas. As such, it is desirable to be able to replicate / predict such decisions when beneficial even in the absence of experts. Unfortunately, there are many obstacles that still hinder an accurate simulation of expert decisions. First, it is hard to understand, and therefore replicate, the way each expert "aggregates" information/assessment along several criteria. In addition, even if we had a reasonable insight about it, any expert may make inconsistent decisions across similar scenarios. Finally, in the case of multiple experts, despite looking at the same information, two (or more) experts may disagree on the decisions to be made.

In spite of such challenges, traditional approaches seek to combine prior known decisions of experts into a classification of scenarios (machine learning approaches) or into some aggregation function that allows to best replicate the experts' decisions. Unfortunately, this line of approaches tends to overlook the irrationality and/or lack of fairness of experts, aggregating all available prior information regardless of quality.

In this work, we propose to model Multi-Experts Multi-Criteria Decision-Making (MEMCDM) problems using argumentation frameworks. We specifically design our proposed model so as to emulate fairness and rationality in decisions. For instance, when, of two expert's decisions, one is unfair, we impose an attack between these two decisions, forcing one of the two decisions out of the argumentation network's resulting extensions. Similarly, we specifically put irrational decisions in opposition to force one out. In doing so, we aim to enable the prediction of decisions that are themselves fair and rational. Our model is illustrated on two toy examples.

In what follows, we start by recalling preliminary notions, then we proceed with describing our model in details and illustrate our model in the case of Software Quality Assessment by multiple experts along multiple criteria.

2 Preliminary Notions

2.1 Multi-Criteria Decision Making (MCDM)

Multi-criteria decision-making (MCDM) involves selecting one of several different alternatives, based on a set of criteria that describe the alternatives. However, there are numerous problems that make comparing these alternatives difficult. For instance, very often, decisions are based on several conflicting criteria; e.g., which car to buy that is cheap and energy efficient. In addition, what happens when we have a group of decision makers that must come to some sort of consensus? This is known as multi-expert multi-criteria decision making (MEMCDM). In MEMCDM, there are several new problems to be addressed. One such problem is how to handle expert disagreement and come to a consensus/decision in the first place. Another problem, as stated earlier, is that of predicting future decisions based on decision data from multiple experts along multiple criteria. Again, the question of "which expert/decision-making process to follow?" is a major challenge in solving such problems.

Approaches to MCDM In general, on a daily basis, when the decision is not critical, in order to reach a decision, we mentally "average / sort" these criteria along with their satisfaction levels. This corresponds to aggregating values of satisfaction with weights on each criterion, reflecting its importance in the overall score (a.k.a. additive aggregation), that is, calculating the overall score of an alternative with the weighted sum of the criterion scores. In other words, weights assigned to different sets of criteria in the weighted-average approach form an "additive measure". Additive aggregation, however, assumes that criteria are independent, which is seldom the case [2]. Non-linear approaches also prove to lead to solutions that are not completely relevant [6].

This should change when considering possible dependence between criteria. For example, if two criteria are strongly dependent, it means that both criteria express, in effect, the same attribute. As a result, when we consider the set consisting of these two criteria, we should assign to this set the same weight as to each of these criteria – and not double the weight as in the weighted sum approach. In general, the weight associated to different sets should be different from the sum of the weights associated to individual criteria. In mathematics, such non-additive functions assigning numbers to sets are known as non-additive (fuzzy) measures. It is therefore reasonable to describe the dependence between different criteria by using an appropriate non-additive (fuzzy) measure. Combining the fuzzy measure values with the criteria satisfaction can be done using the Choquet integral, which integrals are actively used in Multi-Criteria Decision Making [5].

However, to make this happen, fuzzy measures need to be determined: they can either be identified by a decision maker/expert or by an automated system that extracts them from sample data. Since human expertise might not always be available and getting accurate fuzzy values (even from an expert) might be tedious [9], fuzzy measures are usually automatically extracted from prior decision decision data. To the original problem, this approach adds an optimization problem that can be tedious to solve. Although it was solved with success for some data sets [10], the overall prediction quality is not satisfactory and the approach limits the number of criteria that can be taken into account (the number of variables to determine is exponential in the number of criteria) [8].

2.2 Argumentation Frameworks

In this section we briefly summarise the background information related to classical AAFs [4]. We focus on the basic definition of an AAF (see Def. 1), on the notion of defence (Def. 2), and on extension-based semantics (Def. 3).

Definition 1 (Abstract Argumentation Frameworks). *An Abstract Argumentation Framework (AAF) is a pair $F = \langle A, R \rangle$ of a set A of arguments and a binary relation $R \subseteq A \times A$, called the attack relation. $\forall a, b \in A$, aRb (or, $a \rightarrowtail b$) means that a attacks b. An AAF may be represented by a directed graph (an interaction graph) whose nodes are arguments and edges represent the attack relation. A set of arguments $S \subseteq A$ attacks an argument a, i.e., $S \rightarrowtail a$, if a is attacked by an argument of S, i.e., $\exists b \in S. b \rightarrowtail a$.*

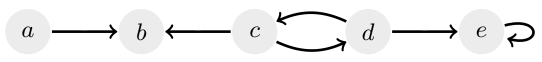


Fig. 1. An example of AAF.

Definition 2 (Defence). *Given an AAF, $F = \langle A, R\rangle$, an argument $a \in A$ is defended (in F) by a set $S \subseteq A$ if for each $b \in A$, such that $b \rightarrowtail a$, also $S \rightarrowtail b$ holds. Moreover, for $S \subseteq A$, we denote by S_R^+ the set $S \cup \{b \mid S \rightarrowtail b\}$.*

The "acceptability" of an argument [4] depends on its membership to some sets, called *extensions*: such extensions need to satisfy the properties required by a given semantics, and they characterise a collective "acceptability". In the following, *stb*, *adm*, *prf*, *gde*, *com*, and *sem*, respectively stand for stable, admissible, preferred, grounded, complete, and semi-stable semantics. The intuition behind these semantics is outside the scope of this work (e.g., see [7, Ch. 3]).

Definition 3 (Semantics). *Let $F = \langle A, R\rangle$ be an AAF. A set $S \subseteq A$ is conflict-free (in F), denoted $S \in cf(F)$, iff there are no $a, b \in S$, such that $(a, b), (b, a) \in R$. For $S \in cf(F)$, it holds that:*

- *$S \in stb(F)$, if foreach $a \in A \backslash S$, $S \rightarrowtail a$, i.e., $S_R^+ = A$;*
- *$S \in adm(F)$, if each $a \in S$ is defended by S;*
- *$S \in prf(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ with $S \subset T$;*
- *$S = gde(F)$ if $S \in com(F)$ and there is no $T \in com(F)$ with $T \subset S$;*
- *$S \in com(F)$, if $S \in adm(F)$ and for each $a \in A$ defended by S, $a \in S$ holds;*
- *$S \in sem(F)$, if $S \in adm(F)$ and there is no $T \in adm(F)$ with $S_R^+ \subset T_R^+$.*

We recall that for each AF, $stb(F) \subseteq sem(F) \subseteq prf(F) \subseteq com(F) \subseteq adm(F)$ holds, and that for each of the considered semantics σ (except stable) $\sigma(F) \neq \emptyset$ always holds. Moreover, in case an AF has at least one stable extension, its stable, and semi-stable extensions coincide. Finally, $gde(F)$ is always unique, and $gde(F) \in com(F)$.

Consider the $F = \langle A, R\rangle$ in Fig. 1, with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. We have that $stb(F) = sem(F) = \{\{a, d\}\}$, and $gde(F) = \{a\}$. The admissible sets of F are $\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}$, and $prf(F) = \{\{a, c\}, \{a, d\}\}$. The complete extensions are $\{a\}, \{a, c\}, \{a, d\}$.

In the proposed model (precisely in Sec. 3.2) we take advantage of *symmetric AAFs* [3]:

Definition 4 (Symmetric AAFs [3]). *A symmetric (Abstract) Argumentation Framework is a finite Argumentation Framework $F = \langle A, R\rangle$ where R is assumed symmetric, non empty and irriflexive.*

This leads to some properties related to the computed semantics: for instance, $\forall S \in prf(F)$ then $S \in stb(F)$, $cf(F) = adm(F)$, and, since each argument in our model is attacked, $gde(F) = \emptyset$ always holds [3]. Note also that in symmetric AAFs, the computation of the sceptical/credulous state (see Def. 5) of an argument becomes easier [3] (e.g., P instead of NP).

Definition 5 (Acceptance state). *Given a semantics σ and a framework F, an argument a is* i) *sceptically accepted iff $\forall S \in \sigma(F), a \in S$, and* ii) *credulously accepted if $\exists S \in \sigma(F), a \in S$.*

Since Argumentation-based decision-making checks the justification state of arguments in order to rank decisions (see a brief summary in the following paragraph), such decision process can benefit from this simplification derived from using symmetric AAFs in our model.

Decision-making with Arguments In this section we simplify part of the content in [7, Ch. 15]. Solving a decision problem amounts to defining a pre-ordering, usually a complete one, on a set $D = \{d_1, \dots, d_n\}$ of n candidate options. Argumentation can be a means for ordering this set D, that is to define a preference relation $\succcurlyeq$ on D. An argumentation-based decision process can be decomposed into the following steps:

1. Constructing arguments in favour/against statements (beliefs or decisions).
2. Evaluating the strength of each argument.
3. Determining the different conflicts among arguments.
4. Evaluating the acceptability of arguments.
5. Comparing decisions on the basis of relevant accepted arguments.

We need to characterise the subsets of practical arguments that are respectively in favour ($\mathcal{F}_f$), or against ($\mathcal{F}_c$) a given option in $d_i \in D$:

- $\mathcal{F}_f : D \to 2^A$ is a function that returns the arguments in favour of a candidate decision. Such arguments are said pros the option.
- $\mathcal{F}_c : D \to 2^A$ is a function that returns the arguments against a candidate decision. Such arguments are said cons the option.

In Def. 6 we present one of the possible ways to prefer ($\succcurlyeq$) one decision instead of another. This *unipolar* principle only refers to either the arguments pros or cons.

Definition 6 (Counting arguments pros/cons). *Let $DS = (D, F)$ be a decision system, where F is an AAF, and $Acc_{stb}(F)$ collects the sceptically accepted arguments of a framework F under the stable semantics. Let $d_1, d_2 \in D$.*

$$d_1 \succcurlyeq d_2 \iff |\mathcal{F}_f(d_1) \cap Acc_{stb}(F)| \geq |\mathcal{F}_f(d_2) \cap Acc_{stb}(F)|$$

The aim of (part of) future work (see also Sec. 5) is to apply similar techniques to derive the best decision about our model, e.g., an evaluation about the software.

3 Proposed Model for MEMCDM using Argumentation Frameworks

Here, we describe our model: given an MEMCDM problem with n criteria and p experts, how do we "translate"/model it as an AAF? In other words, which arguments and attacks should compose it? Note that, through this section we will use letters S and R to identify "Software", "Ranking" (unlikely to Sec. 2.2, where these letters represent a subset of arguments and the attack relation respectively).

3.1 Arguments

• What does the data we use (i.e., experts' evaluation of software in this case) tell us about the arguments to add to the network? We differentiate arguments that come from the data (i.e., Expert i said that Software j is good) from arguments that are implicit (i.e., Software k is Poor).

1. **Expert i gives Item j a total quality** D_{ij} (which, in the case of Software Quality Assessment – SQA, can be Bad, Poor, Fair, Good, or Excellent):

   $$\textbf{Argument} \ \ (E_i, S_j, D_{ij})$$

   Let us call such arguments, arguments of type ESD.

2. **Expert i judges that Item j satisfies criterion m up to quality** D_{ijm}

   $$\textbf{Argument} \ \ (E_i, S_j, c_m, D_{ijm})$$

   Let us call such arguments, arguments of type EScD.

• Which implicit arguments should be part of the argumentation network for this specific type of problem?

1. For each item, independently from what experts say, there will be a decision made. This decision will be in the form of a final ranking, ranging over all possibly ranking values (in the case of SQA: Bad, Poor, Fair, Good, Excellent). So regardless of ESD arguments, we add to the argumentation network the following arguments:

   $$\forall \text{ item } S_i, \forall \text{ ranking } D_j : \ \textbf{Argument} \ \ (S_i, D_j)$$

   Let us call such arguments, **arguments of type SD**.

2. For each criterion of evaluation, regardless of which item is being evaluated and of what experts will decide, a ranking will be associated. So regardless of EScD arguments, we add to the argumentation network the following arguments:

   $$\forall \text{ criterion } c_k, \forall \text{ ranking } D_m : \ \textbf{Argument} \ \ (c_k, D_m)$$

Let us call such arguments, **arguments of type cD**. *Such arguments are expected to be useful for prediction of the decision of experts on items not part of the original data, but for which we do have an indication of their quality per criterion.*

• **Coalitions of Arguments** Here we aim to model the fact the n decisions of any expert on the n criteria of the problem at hand belong together: they together form the "support" for the expert's final decision on the given item. As a result, for any expert E_i and any item S_j, we define a coalition of "supporting" decisions as:

$$\forall E_i, \forall S_j,\ \textbf{Coalition: } \{(E_i, S_j, c_k, D_{i,j,k}),\ k \in \{1, \ldots, n\}\}$$

Let us call such coalitions of EScDs, extended arguments of type **CoEScD**. The result of modeling such coalitions is that all arguments in the coalition will be forced to be altogether either in or out of extensions. *Per se, we are enforcing an equality constraint on the belonging of these arguments to any extension.*

Note that here we do not use the term "support" as in classical *Bipolar AAFs* [1] (or *BAAFs*), which exploit the notion of a *support* binary-relation among arguments. There, the support relation is totally independent of the attack one.

3.2 Attacks

In this subsection, we answer the following question: What are the **attacks** (*edges of the network*) between these arguments (*nodes*)? *Note:* All attacks we define are reciprocal, hence the edges are always set bidirectionally.

For attacks too, we differentiate between attacks that come from inconsistencies in the decision data (disagreement between experts, inconsistency in decisions of a single expert, lack of fairness, irrationality). An assumption that we make in designing the network model is that experts should be rational: in this, we mean that even if they are not (which we know), they should be and we aim to elicit decisions that are as rational as can be.

• **Attacks derived from lack of fairness** Here, we assume that if an expert is fair, then s/he should derive the same final ranking from the same criteria rankings. For instance, if there are 3 criteria (c_1, c_2, and c_3) to assess items and an expert E has the following decision history:

$$\begin{cases} E, S_i, c_1, D_1 \\ E, S_i, c_2, D_1 \\ E, S_i, c_3, D_1 \end{cases} \longrightarrow\ E, S_i, D$$

and: (with $S_i \neq S_j$)

$$\begin{cases} E, S_j, c_1, D_1 \\ E, S_j, c_2, D_1 \\ E, S_j, c_3, D_1 \end{cases} \longrightarrow \ E, S_j, D'$$

where $D \neq D'$, then we should see arguments (E, S_i, D) and (E, S_j, D') are a lack of fairness in judgment and therefore add the following attack in the argumentation network: $(E, S_i, D) \longleftrightarrow (E, S_j, D')$.

More generally, assuming that the criteria that are considered by the experts are c_k, with $k \in K$, and that the possible rankings are denoted by D_r, with $r \in R$, then we add the following rule to our model:

$$\forall E, S_i, S_j, \ \text{ s.t. } i \neq j \text{ and } \forall k \in K, \exists r \in R, \ (E, S_i, c_k, D_r) \text{ and } (E, S_j, c_k, D_r):$$

$$\begin{array}{c} \text{if } (E, S_i, D_i) \text{ and } (E, S_j, D_j) \text{ and } D_i \neq D_j \\ \text{then } \textbf{Attack } (E, S_i, D_i) \longleftrightarrow (E, S_j, D_j) \end{array}$$

• **Attacks derived from lack of rationality** Let us recall that we assume that the rankings D_r, with $r \in R$, are totally ordered. However, with n criteria, the set of n-tuples of rankings is only partially ordered:

$$\begin{array}{l} (D_1, D_2, \ldots, D_n) \ \prec (D'_1, D'_2, \ldots, D'_n) \\ \text{iff :} \\ \forall i \in \{1, \ldots, n\} : \ (D_i \neq D'_i) \ \longrightarrow \ D_i < D'_i \end{array}$$

Now: $\forall E_i$ and $\forall S_j$, we denote by $(D_{1,i,j}, \ldots, D_{n,i,j})$ the set of n decisions made by Expert E_i on each of the criteria $c_1, \ldots, c_n$ for Item S_j, and by $D_{i,j}$ the final decision of Expert E_i on Item S_j.

Being rational for any given expert E_i means that if for Item S_j, s/he ranks criteria lower (w.r.t. above partial order) than s/he ranks the criteria of Item S_k, then his/her final ranking of S_j should not be higher than his/her ranking of S_k. Formally, it is expressed as follows:

$$\begin{array}{l} \forall E_i, \ \forall S_j, \ \forall S_k (j \neq k): \\ \text{if: } (D_{1,i,j}, \ldots, D_{n,i,j}) \prec (D_{1,i,k}, \ldots, D_{n,i,k}) \text{ and: } D_{i,j} > D_{i,k} \\ \text{then: } \textbf{Attack } (S_j, E_i, D_{i,j}) \ \longleftrightarrow \ (S_k, E_i, D_{i,k}) \end{array}$$

• **Attack related to implicit arguments: SD and cD** In this subsection, we describe the following attacks:

– attacks between implicit arguments SD (resp. cD), and
– attacks across SD and ESD (resp. cD and EScD).

1. Attacks among SDs: SD Arguments associate an item with a ranking. For each item S_i, there are p SD arguments if there are p possible ranking levels. Each of these p arguments attack each other (they form a complete subgraph). In other words:

$$\forall S_i, \forall r_1, r_2 \in R, \text{ with } r_1 \neq r_2, \textbf{Attack:}\ (S_i, D_{r_1}) \longleftrightarrow (S_i, D_{r_2})$$

2. Attacks among cDs: In a fashion similar to attacks among SDs, we have:

$$\forall c_j, \forall r_1, r_2 \in R, \text{ with } r_1 \neq r_2, \textbf{Attack:}\ (c_j, D_{r_1}) \longleftrightarrow (c_j, D_{r_2})$$

3. Attacks between SDs and ESDs: For any given item S_j, an argument saying that S_i is evaluated D_h is in contradiction (and therefore attacks – and vice-versa) with any argument (E_i, S_j, D_k) as soon as $D_h \neq D_k$. As a result:

$$\forall E_i,\ \forall S_j,\ (D_h \neq D_k)\ \rightarrow\ \textbf{Attack:}\ (S_j, D_h) \longleftrightarrow (E_i, S_j, D_k)$$

4. Attacks between cDs and EScDs: Similarly as above, for any given criterion c_m, an argument saying that c_m is evaluated D_h is in contradiction (and therefore attacks – and vice-versa) with any argument (E_i, S_j, c_m, D_k) as soon as $D_h \neq D_k$. As a result:

$$\forall E_i,\ \forall S_j,\ \forall c_m, (D_h \neq D_k)\ \rightarrow\ \textbf{Attack:}\ (c_m, D_h) \longleftrightarrow (E_, S_j, c_m, D_k)$$

• **Attacks between Coalitions and ESDs** Here we aim to model the fact that coalitions of decisions on criteria support experts' decisions. In other words:

$$\forall E_i, \forall S_j,\ \{(E_i, S_j, c_k, D_{i,j,k}),\ k \in \{1, \ldots, n\}\}\ \textbf{supports}\ (E_i, S_j, D_{i,j})$$

In terms of attacks, this is expressed as follows:

$$\begin{array}{l}\forall E_i, E_j \forall S_k:\ D_{i,k} \neq D_{j,k}\ \rightarrow \\ \textbf{Attack:}\ \{(E_i, S_k, c_l, D_{i,k,l}),\ k \in \{1, \ldots, n\}\} \longleftrightarrow (E_j, S_k, D_{j,k})\end{array}$$

4 An Example

Here, let us look at a scenario in which experts independently assess given pieces of software, based on several given evaluation criteria. We describe the resulting argumentation networks (arguments/nodes and attack/edges). Table 1 summarises our example, by reporting all the Poor/Fair/Good quality-evaluation about two different criteria (1 and 2) and the overall quality related to three different software products (S1/S2/S3). Such scores are produced by three different experts (E1/E2/E3). For instance, E1 estimates that the overall quality of S1 is fair, with Criterion 1 evaluated as poor, and Criterion 2 as good.

The graph in Fig. 2 represents the AAF given by following the model proposed in Sec. 3 on the data in Tab. 1. The yellow nodes represent explicit arguments from the data. The green nodes are the implicit arguments. The blue nodes are the coalitions. The black bold lines represent attacks due to lack of fairness and lack of rationality. The dotted line attacks are those based on implicit arguments. Finally, the grey bold lines are coalition supports of expert decisions.

Table 1. The explicit arguments that can be collected on our toy-example.

Software	Expert	Criterion 1	Criterion 2	Total Quality
S1	E1	Poor	Good	Fair
S1	E2	Good	Poor	Fair
S1	E3	Fair	Fair	Fair
S2	E1	Poor	Good	Poor
S2	E2	Poor	Good	Good
S2	E3	Poor	Good	Fair
S3	E1	Good	Good	Good
S3	E2	Good	Good	Fair
S3	E3	Fair	Fair	Fair

5 Conclusion and Future Work

In this work, we proposed a model for MEMCDM problems, based on classical AAFs [4], that allows to emulate fairness and rationality. This allows discrimination among input decision data (from experts' prior decisions) between data of value and data that should just not be taken into account. Next steps include operationalising the whole process (from input processing to results filtering) and then adding weights to the attacks to simulate the extent of disagreements and allow lineance towards small errors (e.g., unfairness / irrationality that are really minimal, minor disagreements). Furthermore, we will take inspiration from classical decision-making techniques [7, Ch. 15] with the purpose to rank decisions and decide, for instance, if a software is good or poor. We will even develop new techniques exploring weights on attacks. Also part of future work, we plan to explicitly acknowledge in the AAF that disagreement can be at two different levels: epistemic and pragmatic, and to make use of argumentation frameworks to identify disagreement configurations (epistemic and pragmatic, epistemic only, pragmatic only).

Acknowledgments. S. Bistarelli was partially supported by MIUR-PRIN "Metodi logici per il trattamento dell'informazione". M. Ceberio's work was partially supported by the National Science Foundation, NSF CCF grant 0953339 and the American Association for the Advancement of Science, AAAS MIRC (agreement date 112612). F. Santini was partially supported by MIUR PRIN "Security Horizons".

References

1. L. Amgoud, C. Cayrol, and M.-C. Lagasquie-Schiex. On the bipolarity in argumentation frameworks. In J. P. Delgrande and T. Schaub, editors, *NMR*, pages 1–9, 2004.
2. M. Ceberio and F. Modave. An interval-valued, 2-additive Choquet integral for multi-criteria decision making. In *Proceedings of the 10th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'04)*, Perugia, Italy, July 2004.

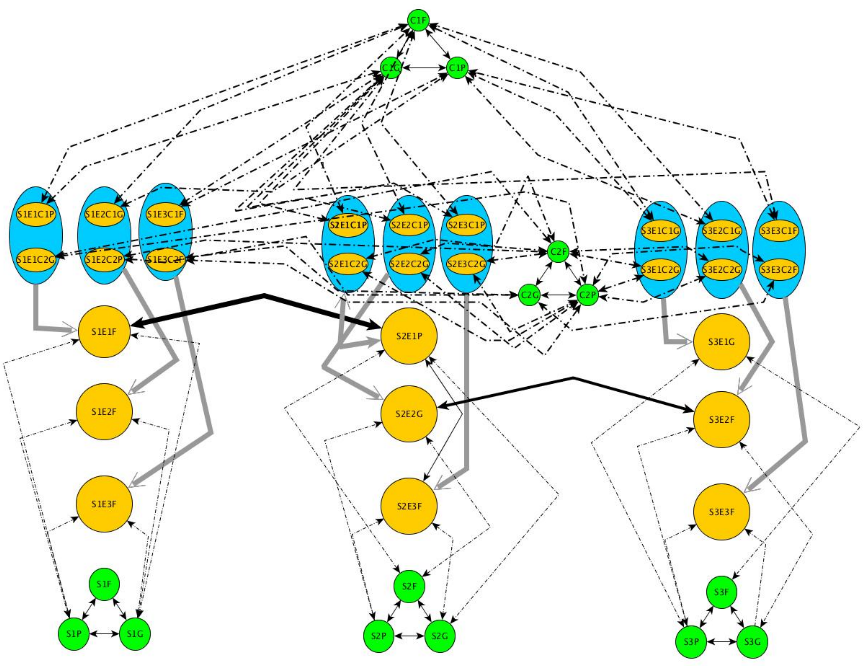


Fig. 2. The AAF given by the model proposed in Sec. 3 on the data in Tab. 1.

3. S. Coste-Marquis, C. Devred, and P. Marquis. Symmetric argumentation frameworks. In L. Godo, editor, *ECSQARU*, volume 3571 of *LNCS*, pages 317–328. Springer, 2005.
4. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–357, 1995.
5. M. Grabisch and C. Labreuche. A decade of application of the choquet and sugeno integrals in multi-criteria decision aid. *4OR*, 6(1):1–44, 2008.
6. F. Modave, M. Ceberio, and V. Kreinovich. Choquet integrals and OWA criteria as a natural (and optimal) next step after linear aggregation: A new general justification. In *Proceedings of MICAI'2008*, pages 741–753, 2008.
7. I. Rahwan and G. R. Simari. *Argumentation in Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2009.
8. X. Wang, M. Ceberio, S. Virani, A. Garcia, and J. Cummins. A hybrid algorithm to extract fuzzy measures for software quality assessment. *Journal of Uncertain Systems*, 7(3):219–237, 2013.
9. X. Wang, A. F. G. Contreras, M. Ceberio, C. D. Hoyo, L. C. Gutierrez, and S. Virani. Interval-based algorithms to extract fuzzy measures for software quality assessment. In *Proceedings of Annual Conference of North American Fuzzy Information Processing Society (NAFIPS'2012)*, Berkeley, CA, August 2012.
10. X. Wang, J. Cummins, and M. Ceberio. The Bees algorithm to extract fuzzy measures for sample data. In *Proceedings of Annual Conference of North American Fuzzy Information Processing Society (NAFIPS'2011)*, El Paso, TX, March 2011.

Optimal placement of storage nodes in a wireless sensor network

Gianlorenzo D'Angelo[1], Daniele Diodati[2], Alfredo Navarra[2], and Cristina M. Pinotti[2]

[1] Gran Sasso Science Institute (GSSI), L'Aquila, Italy.
gianlorenzo.dangelo@gssi.infn.it
[2] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy.
daniele.diodati@dmi.unipg.it; alfredo.navarra@unipg.it; cristina.pinotti@unipg.it

Networks of sensor nodes are usually employed to monitor large areas, collecting data with regular frequency. This large volume of data has to be stored somewhere for answering to external user queries [3]. There are usually two main ways to store data. Source nodes, which are responsible for collecting data, can either locally store the data or transmit them to the *sink*, a powerful node connected to the external world. Both solutions present some disadvantages. If data are locally stored, several problems may arise: (i) data cannot be accumulated for long periods because nodes are equipped with only limited memory space; (ii) stored data are lost once the energy of a source node – battery operated – is depleted; and (iii) searching data for serving query demand results in network-wide communications. Alternatively, source nodes can forward the collected data to the sink. However, communicating data from the source nodes up to the sink makes the network congested, especially if data are transmitted *raw*, that is, uncompressed. Limitations to the number of packets a sensor can transmit to the sink per time unit must be also considered [2].

Recently [9, 10], a hybrid solution has been proposed which makes use of a limited number of "special" sensors, more powerful than standard ones in terms of storage, energy, and computational capabilities. Under this model, source nodes may forward their raw data to such special nodes, referred to as *storage nodes*. Here, raw data are stored and *compressed*, i.e., reduced in size, to be transmitted to the sink at the time a query demand from external users is submitted. With this two-tier model, if the number of storage nodes is kept limited, the network becomes less congested at the price of a moderate increase of the sensor cost of the network. Indeed, the integration of storage nodes in the tiered architecture for sensor networks is made possible by the new storage-enriched hardware [6, 11] considered to be very practical [5]. The introduction of the storage nodes helps to alleviate the transmission bandwidth problem by distributing the local data transmission to the storage nodes. This hierarchical structure has been instantiated by the popular stargate device [11] and the memory-enhanced sensor nodes by UC Riverside [6]. Those special powerful nodes take advantage of their high transmission, storage and even computational capabilities to alleviate the bandwidth limitation, and also provide auxiliary support for surrounding vulnerable sensors for data back-up. In [9, 10] the problem of selecting a subset of storage nodes so as the overall communication cost is minimized is called *optimal storage placement* problem. When the number of storage nodes is limited by an integer k, we talk about the *minimum* k*-storage problem*, which is formally stated in the next paragraph.

Problem statement. Let $G = (V, E)$ be a connected directed graph of n nodes representing a sensor network. Each node $v \in V$ generates raw data of size $s(v)$. Arcs of the network have different weights. The energy cost propagation of a message over the arc $(u, v) \in E$ is denoted by $w(u, v)$. When a communication link is bidirectional, $w(u, v) = w(v, u)$. Let $d(u, v)$ be the minimum energy cost for propagating a message from u to v which is given by the shortest path distance from u to v in G. Each $v \in V$ can be set to serve as *storage* node. A solution is a set $S \subseteq V$ of storage nodes such that $|S| \leq k$, for some $k \in \mathbb{N}$. External users retrieve data from a special storage node $r \in S$, named *sink*. Each node v in V is associated to a storage node in S, denoted as $\sigma(v, S)$. Clearly, if $v \in S$, then $\sigma(v, S) = v$. For replying to a query, a storage node compresses and sends to r the last data generated from its associated nodes. The compressed size of the data produced by a node v becomes $\alpha s(v)$, with $\alpha \in [0, 1]$. The compressed data cannot be further compressed if they reach another storage node. A node $v \in V$ is associated to the storage node $s \in S$ that minimizes $s(v)d(v, s) + \alpha s(v)d(s, r)$, ties are arbitrarily broken. The total cost for a set S of storage nodes is given by: $\mathit{cost}(S) = \sum_{v \in V} s(v) \left(d(v, \sigma(v, S)) + \alpha d(\sigma(v, S), r)\right)$. The *minimum* k*-storage* problem (briefly, *MSP*) consists in finding a subset $S \subseteq V$, with $|S| \leq k$ that minimizes $\mathit{cost}(S)$.

Related Work. There has been a lot of prior research on data collection in sensor networks. Initially, no in-network storage was considered: the request for data was routed from the sink to every sensor by flooding messages. The data were sent to the sink by following the same path but in the reverse direction [4]. Recently, a two-tier model has been proposed [10] to ameliorate the problem of communication congestion. The authors formulate the problem as an integer programming problem and propose a 10-approximation rounding algorithm. Differently from us, they assume that (i) raw data have size independent from the source nodes; and (ii) the energy spent for transmitting one unit of data between any pair of sensors is proportional to their Euclidean distance. For us, instead, different source nodes may generate data of different size, since sensors can monitor different environment aspects. Moreover, we assume that communications follow an underlying network represented by a graph. Each edge of the graph has its own weight that measures the energy required to traverse it. In [9], the problem is solved assuming that the communication network topology is a directed tree T, rooted at the sink. The arcs are directed towards the sink to collect the data, and they are directed away from the sink to broadcast the query. When a sensor s sends one unit data upwards to the sink the energy cost is fixed, while when a sensor s sends one unit data downwards, the energy cost can be high because it is proportional to the number of children of s in T. In this paper and in [10], instead, storage nodes simply send query replies in a proactive manner with a predefined query frequency and hence the query cost is null.

Our results. In the following we summarize our results. In detail we first focus on the case of directed graphs and then in that of undirected ones.

Directed graphs. First, we focus on the approximation properties of the problem and we show that indeed the problem is not in APX, that is, we cannot devise a polynomial time algorithm with a constant factor approximation guarantee. This is stated in the next theorem.

Theorem 1. *Unless $P = NP$, MSP in directed graphs does not belong to APX.*

The above theorem implies that it is not possible to find any practical approximation guarantee for the general case. Therefore, we focus on a restricted case where the topology is a tree rooted at the sink and all the arcs are directed towards the sink. In this case, we show that *MSP* can be optimally solved by a dynamic programming algorithm in $O(\min\{kn^2, k^2P\})$ time, where P is the *path length* of the tree [8].[3] We observe that for a balanced binary tree $P = \Theta(n \log n)$, for random general trees $P = \Theta(n\sqrt{n})$, and in the worst case $P = O(n^2)$.

Undirected graphs. The proof of Theorem 1 is strictly based on the fact that the links of the network are unidirectional and does not hold if all the links of the network are bidirectional, that is the underlying graph is undirected. This is a realistic assumption as links of a sensor network are usually bidirectional.

As the minimum k-storage problem for undirected graphs is similar to the well-known metric k-median problem [1], then it is easy to show that also in this case the problem is *NP*-complete.[4] However, we are able to show that for graphs with bounded treewidth [7], the problem is optimally solvable in polynomial time. We note that this result also holds for the metric k-median problem which is interesting by itself.

Theorem 2. *Given a tree-decomposition of size w, there exists an algorithm that optimally solves MSP in $O(w \cdot k \cdot n^{w+3})$ time.*

Notice that the above theorem does not prove that *MSP* is fixed parameter tractable and we leave such proof (or disproof) as an open problem.

We then characterize the minimum k-storage problem on undirected graphs from the approximation viewpoint. To this aim, we first prove that it is *NP*-hard to approximate the metric k-median problem within a factor of $1 + \frac{1}{e}$, and then we extend such a bound to the minimum k-storage problem by means of a polynomial time reduction that preserves approximation.

[3] The path length of a tree is the sum of the lengths over all nodes of the paths from the root to each node.

[4] The metric k-median problem is defined as follows: Given a complete graph $G = (V, E)$, a metric distance function $dist : V \times V \to \mathbb{N}$, and an integer k, find a set $V' \subseteq V$ such that $|V'| \leq k$ and $\sum_{u \in V} \min_{v \in V'} dist(u, v)$ is minimized.

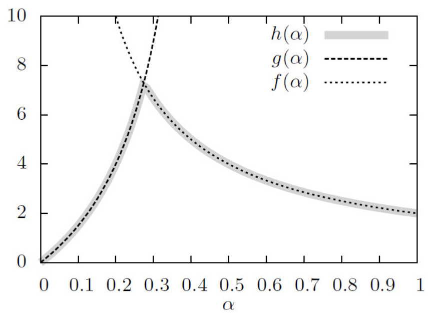


Fig. 1a. The two upper bound functions to the locality gap.

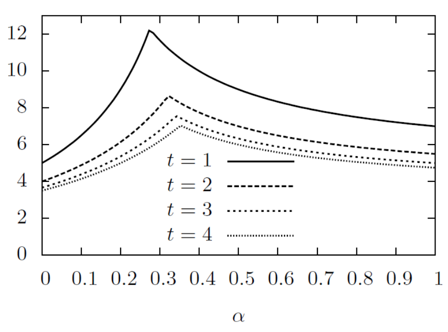


Fig. 1b. Function h'.

Theorem 3. *It is NP-hard to approximate the metric k-median problem within a factor $\gamma < 1 + \frac{1}{e}$.*

Corollary 1. *It is NP-hard to approximate MSP within a factor $\gamma < 1 + \frac{1}{e}$.*

According to Theorem 3, we propose a local search algorithm which guarantees a constant approximation ratio greater than $1+1/e$. In detail, the algorithm is denoted by $\mathcal{L}$ and defined as follows. Each solution is specified by a subset $S \subseteq V$ of exactly k nodes. To move from one feasible solution S to a neighboring one S', we define a *swap* operation between two nodes $s \in S$ and $s' \in V \setminus S$ which consists in adding s' and removing s, that is $S' = S \cup \{s'\} \setminus \{s\}$. In $\mathcal{L}$, we repeatedly check whether any swap move yields a solution of lower cost. In the affirmative case, we apply to the current solution any swap move that improves the solution cost and the resulting solution is set to be the new current solution. This is repeated until, from the current solution, no swap operation decreases the cost, that is, the current solution represents a local optimum. To give a bound on the locality gap, let us define the following three functions: $f : (0,1] \to \mathbb{R}$, $f(\alpha) = 2/\alpha$; $g : [0, \frac{1}{2}) \to \mathbb{R}$, $g(\alpha) = \frac{12\alpha}{1-2\alpha}$; $h : [0,1] \to \mathbb{R}$,

$$h(\alpha) = \begin{cases} g(\alpha) & \text{if } \alpha = 0 \\ \min\{f(\alpha), g(\alpha)\} & \text{if } \alpha \in (0, \frac{1}{2}) \\ f(\alpha) & \text{if } \alpha \in [\frac{1}{2}, 1]. \end{cases}$$

Theorem 4. *The local search algorithm $\mathcal{L}$ for MSP with compression ratio $\alpha \in [0,1]$ exhibits a locality gap of at most $5 + h(\alpha)$.*

Theorem 4 provides two upper bounds to the locality gap given by $5 + f(\alpha)$ and $5+g(\alpha)$. Functions f, g, and h are plotted in Fig. 1a. Function f is monotonic decreasing, while g is monotonic increasing, in their intervals of definition. We have that $f(\alpha) = g(\alpha)$ for $\alpha = \frac{1}{6}(\sqrt{7} - 1) \approx 0.274$ where $f(\alpha) = g(\alpha) < 7.3$. For all the other values of α, one of the two functions is always below such a threshold, that is the approximation ratio is always below 12.3.

Actually, algorithm $\mathcal{L}$ is not yet an approximation algorithm, as the number of iterations needed to find a local optimum solution might be superpolynomial. To fix this problem, as in [1], we can change the stopping condition of $\mathcal{L}$ so it finishes as soon as it finds an approximate local optimum solution, i.e., when the solution S is such that every neighboring solution S' of S has $cost(S') > (1-\epsilon)cost(S)$, for some $\epsilon \in (0,1)$. This leads to the next corollary.

Corollary 2. *There exists an* $\frac{1}{1-\epsilon}(5+h(\alpha))$*-approximation algorithm for MSP for any* $\epsilon \in (0,1)$.

Finally, by following the arguments in [1], the algorithm can be improved by allowing t simultaneous swaps. This leads to a locality gap of $h'(\alpha)$, where

$$h' : [0,1] \to \mathbb{R},\ h'(\alpha) = \begin{cases} g'(\alpha) & \text{if } \alpha = 0 \\ \min\{f'(\alpha), g'(\alpha)\} & \text{if } \alpha \in (0, \frac{t}{t+1}) \\ f'(\alpha) & \text{if } \alpha \in [\frac{t}{t+1}, 1] \end{cases};$$

$f' : (0,1] \to \mathbb{R}$, $f'(\alpha) = 1 + \frac{t+1}{t}\frac{1+2\alpha}{\alpha}$; $g' : [0, \frac{t}{t+1}) \to \mathbb{R}$, $g' = \frac{(3+\alpha)t+2+\alpha}{(1-\alpha)t-\alpha}$.

Function h' is plotted in Fig. 1b for $t = 1,2,3,4$. To give an idea on the improvement provided by this method, we computed the maximum value of the upper bounds on the approximation ratio for $t = 2,3,4$, which is less than 8.67, 7.78 and 7.05, respectively. It follows that for $t \geq 2$ and any value of α, our algorithm improves over the 10-approximation algorithm provided in [10].

References

1. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.
2. E. Duarte-Melo and M. Liu. Data-gathering wireless sensor networks: Organization and capacity. *Computer Networks (COMNET)*, 43(4):519–537, November 2003.
3. J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.
4. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of ACM SIGMOD/PODS Conference*, pages 491–502, 2003.
5. J. Paek, B. Greenstein, O. Gnawali, K.-Y. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, and E. Kohler. The tenet architecture for tiered sensor networks. *ACM Transactions on Sensor Networks*, 6(4):34:1–34:44, 2010.
6. Rise project. http://www.cs.ucr.edu/˜rise, 2014.
7. N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
8. R. Sedgewick and P. Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley, 1996.
9. B. Sheng, Q. Li, and W. Mao. Optimize storage placement in sensor networks. *IEEE Transactions on Mobile Computing*, 9(10):1437–1450, 2010.
10. B. Sheng, C. C. Tan, Q. Li, and W. Mao. An approximation algorithm for data storage placement in sensor networks. In *Proceedings of the 2nd International Conference on Wireless Algorithms, Systems and Applications (WASA)*, pages 71–78. IEEE, 2007.
11. Stargate gateway (spb400). http://www.xbow.com, 2014.

Engineering shortest-path algorithms for dynamic networks

Mattia D'Emidio and Daniele Frigioni

Department of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Via Gronchi 18, I–67100, L'Aquila, Italy.
mattia.demidio@univaq.it, daniele.frigioni@univaq.it

Abstract. The problem of updating shortest paths in networks whose topology dynamically changes over time is a core functionality of many nowadays networked systems. In fact, the problem finds application in many real-world scenarios such as Internet routing and route planning in road networks. In these scenarios, shortest-path data are stored in different ways and have to be updated whenever the underlying graph, representing the network, undergoes dynamic updates. This paper provides a top-level overview of [13], where new dynamic shortest-path algorithms for various real-world applications are proposed, engineered, analyzed and compared to the literature, both theoretically and experimentally.

1 Introduction

In recent years, a massive interest has arisen in the scientific community for new algorithms explicitly designed for nowadays computing systems, such as computer networks, transportation infrastructures and distributed systems. This impulse was motivated by the increasing complexity of such systems, which required new methods and solutions able to overcome the limits of purely theoretical and mathematical approaches to solve problems. In fact, both increasing demand for more efficient solutions to actual real-world problems and advancements in computer hardware, which render traditional computing models more and more unrealistic, have led to a rising gap between classical algorithm theory and algorithmics in practice. The emerging discipline of *algorithm engineering* aims at bridging this gap by complementing theory by the benefits of experimentation. This area of studies has gained even more importance in the last decade, when networked, therefore complex and in most cases dynamic, systems have undergone an astonishing widespread diffusion (see [5, 12, 18, 21]).

In [13] we have focused on engineering new algorithms for the dynamic single-source shortest paths problem, i.e. the problem of computing and updating shortest-path trees in networks whose topology dynamically changes over time. The study was motivated by the importance of this problem, which finds application in many real-world scenarios, such as routing in communication networks and route planning in road networks. In these scenarios, shortest-path data are stored in different ways and need to be updated whenever the underlying graph undergoes dynamic updates. In details, the original contribution of [13] consists of new dynamic shortest-path algorithms for various real-world applications. The work concentrates on problems related to three main categories of networks: *General Networks*, *Communication Networks*, and *Transportation Networks*, respectively. The proposed algorithms are analyzed and compared to the literature, both theoretically and experimentally. In Sections 2–4 we summarize the contributions for each of the aforesaid categories, while Section 5 gives some concluding remarks.

2 General Networks

In this part, we focused on the problem of maintaining the shortest-path tree from a given source of a general graph with positive real edge weights, whose topology undergoes dynamic changes. This problem has been widely studied both theoretically and experimentally. From the theoretical point of view, some solutions have been proposed [14,15,17,20]. Some of them are only able to cope with the update of one edge at a time [14, 15], while others can handle also *batch* updates [17, 20], i.e., updates that consist of multiple edge changes at a time. To the best of our knowledge, none of the above solutions is asymptotically better than recomputing the shortest paths from scratch, by applying Dijkstra's algorithm in the worst case. From the experimental point of view, very few studies are known. The most recent is that in [2], an experimental evaluation of the algorithms in [15, 17, 19, 20] and some of their variants for batch updates. The most important conclusion of this paper is the astonishing level of data dependency within the problem. The second outcome is that it is useful to process a set of updates as a batch when updated edges have strong interference w.r.t. their impact on the shortest-path tree. While updates that are far away from each other usually do not interfere, and hence they can be handled iteratively.

Our contribution to this area is the following: we have developed two new dynamic algorithms for *homogeneous* batches [6], i.e. either *incremental* (containing only insert and weight decrease operations) or *decremental* (containing only delete and weight increase operations) batches which model realistic dynamic scenarios like node failures in communication networks. We have showed that they extend the results of [14] to general graphs, and to batch updates, and those of [15] to batch updates. We have proved the new algorithms to be theoretically efficient in case of homogeneous batches. We have also provided an extensive experimental study that compares the new solutions with the most effective known batch algorithms [7]. Our data show that the proposed algorithms improve over the literature in a set of realistic scenarios. Our results complement previous studies and show that the various solutions can be consistently ranked on the basis of the type of homogeneous batch and of the underlying network.

3 Communication Networks

In this part, we considered the problem of routing in communication networks. The most used approach for solving this problem is that based on shortest paths [4, 16]. If the network is represented by a weighted graph, where vertices model nodes of the network, edges model links connecting such nodes and the weight of an edge models the time required by packets for traversing the corresponding link, the problem can be solved by the distributed computation of all-pairs shortest paths. Known solutions for the problem are usually classified as *distance-vector* and *link-state* [3]. Most distance-vector solutions are based on the classical distributed Bellman-Ford's method and hence converge very slowly, due to well known looping phenomena, but require to store very little data and are able to broadcast information about dynamic changes to a subset of nodes of the network. Link-state algorithms require nodes to store the entire network topology and to compute the shortest path to any destination, usually by running Dijkstra's algorithm, thus requiring quadratic space per node. Link-state algorithms are free of looping, however each node needs to receive and store up-to-date information on the entire network topology after a dynamic change.

In the last decade, there has been a renewed interest in devising new lightweight distributed shortest-path algorithms for a large number of applications where routing devices can have limited storage capabilities, like i.e. wireless sensor networks and large scale ethernet networks. In these applications, loop-free distance-vector algorithms seem to be an attractive alternative to link-state algorithms [21].

Our contribution to this area is twofold. First, we have presented a new loop-free distance-vector algorithm, named *LFR* (Loop Free Routing), which improves over previously known algorithms [9]. From the theoretical point of view, the algorithm has the same message complexity of DUAL [16], one of the best algorithms of the category, but it is always the best choice in terms of memory requirements. From the experimental point of view, LFR outperforms DUAL [9] in terms of messages on a set of real-world networks, whereas DUAL is always the best choice on artificial instances. Second, we have developed a new technique, named *DCP* (Distributed Computation Pruning), which can be combined with every distance-vector algorithm to overcome some of their main limitations (high number of messages sent, high space occupancy per node, low scalability, poor convergence) in power-law networks [10]. We have provided experimental evidences that the use of DCP in combination with DUAL and LFR induces a massive improvement in their performance, in terms of both message complexity and memory requirements.

4 Tranportation Networks

In this part, we studied the problem of computing best connections in transportation networks. The main effort of the study was dedicated to best connections in road graphs, where vertices represent points on a map, edges represent road segments connecting such points, and travel times for each segment are assigned to the corresponding edge. This problem is a variant of the single-source shortest paths problem and hence can be solved by applying Dijkstra's algorithm. Unfortunately, real-world transportation networks tend in general to be huge, yielding unsustainable times to compute shortest paths by traditional approaches. For this reason, many efforts have been done in the last years to accelerate the practical performance of Dijkstra's algorithm on typical instances of road networks. These research efforts have led to the development of a number of so-called *speed-up techniques*, which compute additional data in a preprocessing phase in order to accelerate the answer to shortest-path queries in an on-line phase. Theoretically, none of such speed-up techniques is better than Dijkstra's in the worst case, while, in practice, some of them have been shown to be very effective. For a comprehensive survey we refer to [1].

The main drawback of these techniques is that, in general, they do not work well in dynamic scenarios, when edge weight changes occur to the network due to, e.g., traffic jams. These scenarios are, of course, interesting, as they arise frequently in practice. In order to keep shortest-path queries correct, the preprocessed data need to be updated. The easiest way is to recompute the data from scratch after each change. This is in general unfeasible, as even the fastest methods need too much time. Therefore, in recent years some techniques for updating shortest paths in dynamic scenarios [11, 22] have been developed.

In our work, we focused on the speed-up technique named Arc-Flags and proposed a new approach to efficiently use Arc-Flags in dynamic networks [8]. The new approach consists of a new data structure and a new fully dynamic algorithm. Our study was motivated by the fact that some of the best performing speed-up techniques, such as for example, CHASE and TNR+AF [1], rely on the correctness and the performance of Arc-Flags. Hence, our dynamization of Arc-Flags represents a first step toward the dynamization of these techniques. We provided both theoretical and experimental evidences that confirm this statement. In detail, our study shows that our dynamic approach overcomes previous methods for maintaining the Arc-Flags in dynamic networks.

5 Conclusion

We have proposed a top-level overview of [13], which contains novel contributions in the area of algorithm engineering. In details, we have given, analyzed and compared to the literature, new dynamic shortest-path algorithms for various real-world applications. The new algorithms improve over known approaches in many interesting scenarios and represent a step forward in the development of more efficient shortest-path solutions for dynamic networks.

References

1. R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. *ACM Journal on Experimental Algorithmics*, 15:Article 2.3, 2010.
2. R. Bauer and D. Wagner. Batch dynamic single-source shortest-path algorithms: An experimental study. In *8th International Symposium on Experimental Algorithms (SEA 2009)*, volume 5526 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2009.

3. D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall International, 1992.
4. S. Cicerone, G. D'Angelo, G. Di Stefano, and D. Frigioni. Partially dynamic efficient algorithms for distributed shortest paths. *Theoretical Computer Science*, 411:1013–1037, 2010.
5. S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and V. Maurizio. Engineering a new algorithm for distributed shortest paths on dynamic networks. *Algorithmica*, 66(1):51–86, 2013.
6. A. D'Andrea, M. D'Emidio, D. Frigioni, S. Leucci, and G. Proietti. Dynamically maintaining shortest path trees under batches of updates. In *20th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2013)*, volume 8179, pages 286–297. Springer, 2013.
7. A. D'Andrea, M. D'Emidio, D. Frigioni, S. Leucci, and G. Proietti. Experimental evaluation of dynamic shortest path tree algorithms on homogeneous batches. In *13th International Symposium on Experimental Algorithms*, volume 8504 of *Lecture Notes in Computer Science*, pages 283–294. Springer, 2014.
8. G. D'Angelo, M. D'Emidio, and D. Frigioni. Fully dynamic update of Arc-Flags. *Networks*, (63):283–294, 2014.
9. G. D'Angelo, M. D'Emidio, and D. Frigioni. A loop-free shortest-path routing algorithm for dynamic networks. *Theoretical Computer Science*, 516:1–19, 2014.
10. G. D'Angelo, M. D'Emidio, D. Frigioni, and D. Romano. Enhancing the computation of distributed shortest paths on real dynamic networks. In *1st Mediterranean Conference on Algorithms (MEDALG 2012)*, volume 7659 of *Lecture Notes in Computer Science*, pages 148–158, 2012.
11. D. Delling, A. V. Goldberg, T. Pajor, and R. F. F. Werneck. Customizable route planning. In *10th International Symposium on Experimental Algorithms (SEA 2011)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387, 2011.
12. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.
13. M. D'Emidio. Engineering shortest-path algorithms for dynamic networks. Ph.D. Thesis, University of L'Aquila, Advisor: Prof. Daniele Frigioni, 2014.
14. D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Semi-dynamic algorithms for maintaining single source shortest path trees. *Algorithmica*, 22(3):250–274, 1998.
15. D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251–281, 2000.
16. J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Trans. on Networking*, 1(1):130–141, 1993.
17. P. Narváez, K.-Y. Siu, and H.-Y. Tzeng. New dynamic algorithms for shortest path tree computation. *IEEE/ACM Trans. on Networking*, 8(6):734–746, 2000.
18. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12:2.4:1–2.4:39, 2008.
19. G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problem. *Theoretical Computer Science*, 158:233–277, 1996.
20. G. Ramalingam and T. W. Reps. An incremental algorithm for a generalization of the shortest paths problem. *Journal of Algorithms*, 21:267–305, 1996.
21. S. Ray, R. Guérin, K.-W. Kwong, and R. Sofia. Always acyclic distributed path computation. *IEEE/ACM Trans. on Networking*, 18(1):307–319, 2010.
22. P. Sanders and D. Schultes. Dynamic Highway-Node Routing. In *Proc. of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *Lecture Notes in Computer Science*, pages 66–79, 2007.

Minimal models for rational closure in $\mathcal{SHIQ}$

Laura Giordano[1], Valentina Gliozzi[2], Nicola Olivetti[3], and Gian Luca Pozzato[2] [⋆]

[1] DISIT - Univ. Piemonte Orientale, Alessandria, Italy - laura@mfn.unipmn.it
[2] Dip. di Informatica - Univ. di Torino, Italy - {gliozzi,pozzato}@di.unito.it
[3] Aix-Marseille Université, CNRS, France nicola.olivetti@univ-amu.fr

Abstract. We introduce a notion of rational closure for the logic $\mathcal{SHIQ}$ based on the well-known rational closure by Lehmann and Magidor [21]. We provide a semantic characterization of rational closure in $\mathcal{SHIQ}$ in terms of a preferential semantics, based on a finite rank characterization of minimal models.

1 Introduction

The growing interest of defeasible inference in ontology languages has led, in the last years, to the definition of many non-monotonic extensions of Description Logics (DLs) [23, 11, 19, 2, 20]. The best known semantics for nonmonotonic reasoning have been used to the purpose, from default logic [1], to circumscription [2], to Lifschitz's logic MKNF [10, 22], to preferential reasoning [4, 15], and to rational closure [5].

In this work, we focus on rational closure and, specifically, on the rational closure for $\mathcal{SHIQ}$. Rational closure provides a significant and reasonable nonmonotonic inference mechanism for DLs, still remaining computationally inexpensive. As shown for $\mathcal{ALC}$ in [5], its complexity can be expected not to exceed the one of the underlying monotonic DL. This is a striking difference with most of the other approaches to nonmonotonic reasoning in DLs mentioned above, with the exception of some of them, such as [22, 20]. In particular, we define a rational closure for the logic $\mathcal{SHIQ}$ building on the notion of rational closure in [21] for propositional logic. This is a difference with respect to the rational closure construction introduced in [6] for $\mathcal{ALC}$, which is more similar to the one by Freund [12]. We provide a semantic characterization of rational closure in $\mathcal{SHIQ}$ in terms of a preferential semantics, generalizing to $\mathcal{SHIQ}$ the results for rational closure for $\mathcal{ALC}$ in [16]. This generalization is not trivial, since $\mathcal{SHIQ}$ lacks a crucial property of $\mathcal{ALC}$, the finite model property. Our construction exploits an extension of $\mathcal{SHIQ}$ with a typicality operator $\mathbf{T}$, that selects the most typical instances of a concept C, thus allowing defeasible inclusions of the form $\mathbf{T}(C) \sqsubseteq D$ (the typical Cs are Ds) together with the standard (strict) inclusions $C \sqsubseteq D$ (all the Cs are Ds).

We define a *minimal model semantics* and a notion of minimal entailment for the resulting logic, $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$, and we show that the inclusions belonging to the rational closure of a TBox are those minimally entailed by the TBox, when restricting to *canonical* models. This result exploits a characterization of minimal models, showing that we can

[⋆] G. L. Pozzato is partially supported by the project ODIATI#1 "Ontologie, DIAgnosi e TIpicalità nelle logiche descrittive" of the local research funds 2013 by the Università degli Studi di Torino - part B, supporting young researchers.

restrict to models with finite ranks. We can show that the rational closure construction of a TBox can be done exploiting entailment in $\mathcal{SHIQ}$, without requiring to reason in $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$, and that the problem of deciding if an inclusion belongs to the rational closure of a TBox is EXPTIME-complete. This abstract is based on the full paper [17].

2 A nonmonotonic extension of $\mathcal{SHIQ}$

Following the approach in [13, 15], we define an extension, $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$, of the logic $\mathcal{SHIQ}$ [18] introducing a typicality operator $\mathbf{T}$ to distinguish defeasible inclusions of the form $\mathbf{T}(C) \sqsubseteq D$, defining the (defeasible) properties of typical instances of C, from strict properties of all instances of C ($C \sqsubseteq D$).

We consider an alphabet of concept names $\mathcal{C}$, role names $\mathcal{R}$, transitive roles $\mathcal{R}^+ \subseteq \mathcal{R}$, and individual constants $\mathcal{O}$. Given $A \in \mathcal{C}$, $R \in \mathcal{R}$, and $n \in \mathbb{N}$ we define:
$C_R := A \mid \top \mid \bot \mid \neg C_R \mid C_R \sqcap C_R \mid C_R \sqcup C_R \mid \forall S.C_R \mid \exists S.C_R \mid (\geq nS.C_R) \mid (\leq nS.C_R)$
$C_L := C_R \mid \mathbf{T}(C_R)$ $\qquad S := R \mid R^-$
As usual, we assume that transitive roles cannot be used in number restrictions [18]. A KB is a pair (TBox, ABox). TBox contains a finite set of concept inclusions $C_L \sqsubseteq C_R$ and role inclusions $R \sqsubseteq S$. ABox contains assertions of the form $C_L(a)$ and $S(a, b)$, where $a, b \in \mathcal{O}$.

The semantics of $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ is formulated in terms of rational models: ordinary models of $\mathcal{SHIQ}$ are equipped with a *preference relation* $<$ on the domain, whose intuitive meaning is to compare the "typicality" of domain elements, that is to say $x < y$ means that x is more typical than y. Typical members of a concept C, that is members of $\mathbf{T}(C)$, are the members x of C that are minimal with respect to this preference relation (s.t. there is no other member of C more typical than x).

Definition 1 (Semantics of $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$). *A model $\mathcal{M}$ of $\mathcal{SHIQ}^{R}\mathbf{T}$ is any structure $\langle \Delta, <, I \rangle$ where: Δ is the domain; $<$ is an irreflexive, transitive, well-founded, and modular (for all x, y, z in Δ, if $x < y$ then either $x < z$ or $z < y$) relation over Δ; I is the extension function that maps each concept C to $C^I \subseteq \Delta$, and each role R to $R^I \subseteq \Delta^I \times \Delta^I$. For concepts of $\mathcal{SHIQ}$, C^I is defined as usual. For the $\mathbf{T}$ operator, we have $(\mathbf{T}(C))^I = Min_<(C^I)$, where $Min_<(S) = \{u : u \in S$ and $\nexists z \in S$ s.t. $z < u\}$.*

$\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ models can be equivalently defined by postulating the existence of a function $k_{\mathcal{M}} : \Delta \longmapsto Ord$, and then letting $x < y$ if and only if $k_{\mathcal{M}}(x) < k_{\mathcal{M}}(y)$. We call $k_{\mathcal{M}}(x)$ *the rank of element* x in $\mathcal{M}$. The rank $k_{\mathcal{M}}(x)$ can be understood as the maximal length of a chain $x_0 < \cdots < x$ from x to a minimal x_0 (s.t. for no x', $x' < x_0$). Observe that because of modularity all chains have the same length.

Definition 2 (Model satisfying a knowledge base). *Given a $\mathcal{SHIQ}^{R}\mathbf{T}$ model $\mathcal{M}$= $\langle \Delta, <, I \rangle$, we say that: - a model $\mathcal{M}$ satisfies an inclusion $C \sqsubseteq D$ if $C^I \subseteq D^I$; similarly for role inclusions; - $\mathcal{M}$ satisfies an assertion $C(a)$ if $a^I \in C^I$; and $\mathcal{M}$ satisfies an assertion $R(a, b)$ if $(a^I, b^I) \in R^I$. Given a KB=(TBox,ABox), we say that: $\mathcal{M}$ satisfies TBox if $\mathcal{M}$ satisfies all inclusions in TBox; $\mathcal{M}$ satisfies ABox if $\mathcal{M}$ satisfies all assertions in ABox; $\mathcal{M}$ satisfies KB if it satisfies both its TBox and its ABox.*

Given a KB, we say that an inclusion $C_L \sqsubseteq C_R$ is derivable from KB, written KB $\models_{\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}} C_L \sqsubseteq C_R$, if ${C_L}^I \subseteq {C_R}^I$ holds in all models $\mathcal{M} = \langle \Delta, <, I \rangle$ satisfying KB; similarly for role inclusions. We also say that an assertion $C_L(a)$, with $a \in \mathcal{O}$, is derivable from KB, written KB $\models_{\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}} C_L(a)$, if $a^I \in {C_L}^I$ holds in all models $\mathcal{M} = \langle \Delta, <, I \rangle$ satisfying KB.

Given a model $\mathcal{M} = \langle \Delta, <, I \rangle$, we define the *rank* $k_{\mathcal{M}}(C_R)$ of a concept C_R in the model $\mathcal{M}$ as $k_{\mathcal{M}}(C_R) = min\{k_{\mathcal{M}}(x) \mid x \in {C_R}^I\}$. If ${C_R}^I = \emptyset$, then C_R has no rank and we write $k_{\mathcal{M}}(C_R) = \infty$. It is immediate to verify that:

Proposition 1. *For any $\mathcal{M} = \langle \Delta, <, I \rangle$, we have that $\mathcal{M}$ satisfies $\mathbf{T}(C) \sqsubseteq D$ if and only if $k_{\mathcal{M}}(C \sqcap D) < k_{\mathcal{M}}(C \sqcap \neg D)$.*

The typicality operator $\mathbf{T}$ itself is nonmonotonic, i.e., $\mathbf{T}(C) \sqsubseteq D$ does not imply $\mathbf{T}(C \sqcap E) \sqsubseteq D$. This nonmonotonicity of $\mathbf{T}$ allows us to express the properties that hold for the typical instances of a class (not only the properties that hold for all the members of the class). However, the logic $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ is monotonic: what is inferred from KB can still be inferred from any KB' with KB $\subseteq$ KB'. This is a clear limitation in DLs. As a consequence of the monotonicity of $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$, one cannot deal with irrelevance. For instance, one cannot derive from KB= $\{ \mathit{VIP} \sqsubseteq \mathit{Person}, \mathbf{T}(\mathit{Person}) \sqsubseteq\ \leq 1\ \mathit{HasMarried.Person}, \mathbf{T}(\mathit{VIP}) \sqsubseteq\ \geq 2\ \mathit{HasMarried.Person}\}$ that KB $\models_{\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}}$ $\mathbf{T}(\mathit{VIP} \sqcap \mathit{Tall}) \sqsubseteq\ \geq 2\ \mathit{HasMarried.Person}$, even if the property of being tall is irrelevant with respect to the number of marriages.

In order to overcome this weakness, we strengthen the semantics of $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ by defining a minimal models mechanism which is similar, in spirit, to circumscription. Given a KB, the idea is to: 1. define a *preference relation* among $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ models, giving preference to the model in which domain elements have a lower rank; 2. restrict entailment to *minimal* $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ models (w.r.t. the above preference relation) of KB.

Definition 3 (Minimal models). *Given $\mathcal{M} = \langle \Delta, <, I \rangle$ and $\mathcal{M}' = \langle \Delta', <', I' \rangle$, $\mathcal{M}$ is preferred to $\mathcal{M}'$ ($\mathcal{M} <_{FIMS} \mathcal{M}'$) if (i) $\Delta = \Delta'$, (ii) $C^I = C^{I'}$ for all concepts C, and (iii) for all $x \in \Delta$, $k_{\mathcal{M}}(x) \leq k_{\mathcal{M}'}(x)$ whereas there is $y \in \Delta$ s.t. $k_{\mathcal{M}}(y) < k_{\mathcal{M}'}(y)$. Given a KB, we say that $\mathcal{M}$ is a minimal model of KB w.r.t. $<_{FIMS}$ if it is a model satisfying KB and there is no $\mathcal{M}'$ model satisfying KB s.t. $\mathcal{M}' <_{FIMS} \mathcal{M}$.*

Differently from [15], the notion of minimality here is based on the minimization of the ranks of the worlds, rather then on the minimization of formulas of a specific kind. It can be proved that a consistent KB has at least one minimal model and that satisfiability in $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ is in EXPTIME such as satisfiability in $\mathcal{SHIQ}$.

The logic $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$, as well as the underlying logic $\mathcal{SHIQ}$, does not enjoy the finite model property. However, we can prove that in any minimal model the *rank* of each domain element is finite, which is essential for establishing a correspondence between the minimal model semantics of a KB and its rational closure. From now on, we can assume that the ranking function assigns to each domain element in Δ a natural number.

3 Rational Closure for $\mathcal{SHIQ}$

In this section, we extend to Description Logics the notion of rational closure proposed by Lehmann and Magidor [21] for the propositional case. Given the typicality operator, the typicality assertion $\mathbf{T}(C) \sqsubseteq D$ plays the role of the conditional assertion $C \mid\sim D$ in Lehmann and Magidor's rational logic R.

Definition 4 (Exceptionality of concepts and inclusions). *Let T_B be a TBox and C a concept. C is said to be* exceptional *for T_B if and only if $T_B \models_{\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}} \mathbf{T}(\top) \sqsubseteq \neg C$. A $\mathbf{T}$-inclusion $\mathbf{T}(C) \sqsubseteq D$ is exceptional for T_B if C is exceptional for T_B. The set of $\mathbf{T}$-inclusions of T_B which are exceptional in T_B will be denoted as $\mathcal{E}(T_B)$.*

Given a DL KB=(TBox,ABox), it is possible to define a sequence of non increasing subsets of TBox $E_0 \supseteq E_1 \supseteq E_2 \supseteq \dots$ by letting $E_0 =$ TBox and, for $i > 0$, $E_i = \mathcal{E}(E_{i-1}) \cup \{C \sqsubseteq D \in$ TBox s.t. $\mathbf{T}$ does not occurr in $C\}$. Observe that, being KB finite, there is an $n \geq 0$ such that, for all $m > n, E_m = E_n$ or $E_m = \emptyset$. The definition of the E_i's is similar the definition of the C_i's in Lehmann and Magidor's rational closure [21] except for the addition of strict inclusions.

Definition 5 (Rank of a concept). *A concept C has* rank *i ($rank(C) = i$) for KB=(TBox, ABox), iff i is the least natural number for which C is not exceptional for E_i. If C is exceptional for all E_i then $rank(C) = \infty$, and we say that C has no rank.*

The notion of rank of a formula allows us to define the rational closure of the TBox of a KB. We write KB $\models_{\mathcal{SHIQ}} F$ to mean that F holds in all models of $\mathcal{SHIQ}$.

Definition 6 (Rational closure of TBox). *Let KB=(TBox,ABox). We define, $\overline{TBox}$, the* rational closure of TBox, *as $\overline{TBox} = \{\mathbf{T}(C) \sqsubseteq D \mid$ either $rank(C) < rank(C \sqcap \neg D)$ or $rank(C) = \infty\} \cup \{C \sqsubseteq D \mid$ KB $\models_{\mathcal{SHIQ}} C \sqsubseteq D\}$.*

The rational closure of TBox is a nonmonotonic strengthening of $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ which allows us to deal with irrelevance, as the following example shows. Let TBox = $\{\mathbf{T}(Actor) \sqsubseteq Charming\}$. It can be verified that $\mathbf{T}(Actor \sqcap Comic) \sqsubseteq Charming \in \overline{TBox}$. This nonmonotonic inference does no longer follow if we discover that indeed comic actors are not charming (and in this respect are untypical actors): indeed given TBox′= TBox $\cup$ $\{\mathbf{T}(Actor \sqcap Comic) \sqsubseteq \neg Charming\}$, we have that $\mathbf{T}(Actor \sqcap Comic) \sqsubseteq Charming \not\in \overline{TBox'}$. Also, as for the propositional case, rational closure is closed under rational monotonicity: from $\mathbf{T}(Actor) \sqsubseteq Charming \in \overline{TBox}$ and $\mathbf{T}(Actor) \sqsubseteq Bold \not\in \overline{TBox}$ it follows that $\mathbf{T}(Actor \sqcap \neg Bold) \sqsubseteq Charming \in \overline{TBox}$.

Theorem 1 (Complexity of rational closure over TBox). *Given a TBox, the problem of deciding whether $\mathbf{T}(C) \sqsubseteq D \in \overline{TBox}$ is in* EXPTIME.

The proof of this result in [17] shows that the rational closure of a TBox can be computed using entailment in $\mathcal{SHIQ}$, through a linear encoding of $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ entailment. EXPTIME-completeness follows from the EXPTIME-hardness result for $\mathcal{SHIQ}$ [18].

4 A Minimal Model Semantics for Rational Closure in $\mathcal{SHIQ}$

To provide a semantic characterization of this notion, we define a special class of minimal models, exploiting the fact that in all minimal $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ models the *rank* of each domain element is always finite. First of all, we observe that the minimal model semantics in Definition 3 as it is cannot capture the rational closure of a TBox.

Consider the TBox containing: $\mathit{VIP} \sqsubseteq \mathit{Person}$, $\mathbf{T}(\mathit{Person}) \sqsubseteq \leq 1\ \mathit{HasMarried}.\mathit{Person}$, $\mathbf{T}(\mathit{VIP}) \sqsubseteq \geq 2\ \mathit{HasMarried.Person}$. We observe that $\mathbf{T}(\mathit{VIP} \sqcap \mathit{Tall}) \sqsubseteq \geq 2\ \mathit{HasMarried.Person}$ does not hold in all minimal $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ models of KB w.r.t. Definition 3. Indeed there can be a model $\mathcal{M} = \langle \Delta, <, I\rangle$ in which $\Delta = \{x, y, z\}$, $\mathit{VIP}^I = \{x, y\}$, $\mathit{Person}^I = \{x, y, z\}$, $(\leq 1\ \mathit{HasMarried.Person})^I = \{x, z\}$, $(\geq 2\ \mathit{HasMarried.Person})^I = \{y\}$, $\mathit{Tall}^I = \{x\}$, and $z < y < x$. $\mathcal{M}$ is a model of KB, and it is minimal. Also, x is a typical tallVIP in $\mathcal{M}$ and has no more than one spouse, therefore $\mathbf{T}(\mathit{VIP} \sqcap \mathit{Tall}) \sqsubseteq \geq 2\ \mathit{HasMarried.Person}$ does not hold in $\mathcal{M}$. On the contrary, it can be verified that $\mathbf{T}(\mathit{VIP} \sqcap \mathit{Tall}) \sqsubseteq \geq 2\ \mathit{HasMarried.Person} \in \overline{\mathit{TBox}}$.

Things change if we consider the minimal models semantics applied to models that contain a domain element for *each combination of concepts consistent with KB*. We call these models *canonical models*. Let $\mathcal{S}$ be the set of all the concepts (and subconcepts) occurring in KB or in the query F together with their complements.

Definition 7 (Canonical model). *Given KB=(TBox,ABox) and a query F, a model $\mathcal{M} = \langle \Delta, <, I\rangle$ satisfying KB is* canonical with respect to $\mathcal{S}$ *if it contains at least a domain element $x \in \Delta$ s.t. $x \in (C_1 \sqcap C_2 \sqcap \dots \sqcap C_n)^I$, for each set of concepts $\{C_1, C_2, \dots, C_n\} \subseteq \mathcal{S}$ consistent with KB, i.e. KB $\not\models_{\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}} C_1 \sqcap C_2 \sqcap \dots \sqcap C_n \sqsubseteq \bot$.*

In order to semantically characterize the rational closure of a $\mathcal{SHIQ}^{\mathsf{R}}\mathbf{T}$ KB, we restrict our attention to *minimal canonical models*. Existence of minimal canonical models can be proved for any (finite) satisfiable KB. Let us first introduce the following proposition, which defines a correspondence between the rank of a formula in the rational closure and the rank of a formula in a model (the proof is by induction on the rank i):

Proposition 2. *Given KB and $\mathcal{S}$, for all $C \in \mathcal{S}$, if $\mathit{rank}(C) = i$, then: 1. there is a $\{C_1 \dots C_n\} \subseteq \mathcal{S}$ maximal and consistent with KB such that $C \in \{C_1 \dots C_n\}$ and $\mathit{rank}(C_1 \sqcap \dots \sqcap C_n) = i$; 2. for any $\mathcal{M}$ minimal canonical model of KB, $k_{\mathcal{M}}(C) = i$.*

The following theorem follows from the propositions above:

Theorem 2. *Let KB=(TBox,ABox) be a knowledge base and $C \sqsubseteq D$ a query. We have that $C \sqsubseteq D \in \overline{\mathit{TBox}}$ if and only if $C \sqsubseteq D$ holds in all minimal canonical models of KB with respect to $\mathcal{S}$.*

5 Conclusions and Related Work

In this work we have proposed an extension of the rational closure defined by Lehmann and Magidor to the Description Logic $\mathcal{SHIQ}$, taking into account both TBox reasoning (ABox reasoning is addressed in [17]). There is a number of closely related proposals.

In [13, 15] nonmonotonic extensions of $\mathcal{ALC}$ with the typicality operator $\mathbf{T}$ have been proposed, whose semantics of $\mathbf{T}$ is based on the preferential logic P. The notion of minimal model adopted here is completely independent from the language and is determined only by the relational structure of models.

The first notion of rational closure for DLs was defined by Casini and Straccia in [5], based on the construction proposed by Freund [12] for propositional logic. In [6] a semantic characterization of a variant of the rational closure in [5] has been presented, generalizing to $\mathcal{ALC}$ the notion of minimally ranked models for propositional logic in [14]. Experimental results in [7] show that, from a performance perspective, it is practical to use rational closure as defined in [6]. The major difference of our construction with those is [5, 6] is in the notion of exceptionality: our definition exploits preferential entailment, while [5, 6] directly use entailment in $\mathcal{ALC}$ over a materialization of the KB. In [17] we have shown that our notion of rational closure for the TBox can nevertheless be computed in $\mathcal{SHIQ}$ by exploiting a linear encoding in $\mathcal{SHIQ}$.

The rational closure construction in itself can be applied to any description logic. As future work, we aim to extend it and its semantic characterization to stronger logics, such as $\mathcal{SHOIQ}$, for which the correspondence between the rational closure and the minimal canonical model semantics of the previous sections cannot be established straightforwardly, due to the interaction of nominals with number restrictions. Also, we aim to consider a finer semantics where models are equipped with several preference relations; in such a semantics it might be possible to relativize the notion of typicality, whence to reason about typical properties independently from each other. The aim is to overcome some limitations of rational closure, as done in [8] by combining rational closure and *Defeasible Inheritance Networks* or in [9] with the lexicographic closure.

References

1. Baader, F., Hollunder, B.: Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. J. Autom. Reasoning 15(1), 41–68 (1995)
2. Bonatti, P.A., Lutz, C., Wolter, F.: The Complexity of Circumscription in DLs. Journal of Artificial Intelligence Research (JAIR) 35, 717–773 (2009)
3. Booth, R., Paris, J.: A note on the rational closure of knowledge bases with both positive and negative knowledge. Journal of Logic, Language and Information 7, 165–190 (1998)
4. Britz, K., Heidema, J., Meyer, T.: Semantic preferential subsumption. In: Brewka, G., Lang, J. (eds.) Principles of Knowledge Representation and Reasoning: Proc. KR 2008, pp. 476–484.
5. Casini, G., Straccia, U.: Rational Closure for Defeasible Description Logics. In: Janhunen, T., Niemelä, I. (eds.) Proc. of JELIA 2010. LNAI, vol. 6341, pp. 77–90.
6. Casini, G., Meyer, T., Varzinczak, I.J., Moodley, K.: Nonmonotonic Reasoning in Description Logics: Rational Closure for the ABox. In: DL 2013, CEUR W. Proc. 1014, pp. 600–615.
7. Casini, G., Meyer, T., Moodley, K., Varzinczak, I.J.: Towards Reasoning Practical Defeasible Reasoning in Description Logics. In: DL 2013, 26th Int. Workshop on Description Logics. CEUR Workshop Proceedings, vol. 1014. CEUR-WS.org (2013)
8. Casini, G., Straccia, U.: Defeasible Inheritance-Based Description Logics. In: Walsh, T. (ed.) Proc. of the 22nd Int. Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 813–818.
9. Casini, G., Straccia, U.: Lexicographic Closure for Defeasible Description Logics. In Proc. of Australasian Ontology Workshop, volume 969, 2012.
10. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Transactions on Computational Logic (ToCL) 3(2), 177–225 (2002)
11. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining Answer Set Programming with Description Logics for the Semantic Web. In: Dubois, D., Welty, C., Williams, M. (eds.) Principles of Knowledge Representation and Reasoning: Proc. of KR 2004, pp. 141–151.

12. Freund, M.: Preferential reasoning in the perspective of poole default logic. Artif. Intell. 98(1-2), 209–235 (1998)
13. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: ALC+T: a preferential extension of Description Logics. Fundamenta Informaticae 96, 1–32 (2009)
14. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A minimal model semantics for non-monotonic reasoning. In: Luis Fariñas del Cerro, Andreas Herzig, J.M. (ed.) Proc. of JELIA 2012. LNAI, vol. 7519, pp. 228–241. Springer-Verlag, Toulouse, France (Sptember 2012)
15. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A NonMonotonic Description Logic for Reasoning About Typicality. Artificial Intelligence 195, 165–202 (2013)
16. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Minimal Model Semantics and Rational Closure in Description Logics . In: Eiter, T., Glim, B., Kazakov, Y., Krtzsch, M. (eds.) Proc. of Description Logics (DL 2013). CEUR Workshop Proc., vol. 1014, pp. 168 – 180.
17. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Rational Closure in $\mathcal{SHIQ}$. In: DL 2014, 27th International Workshop on Description Logics. To appear (2014)
18. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL 8(3), 239–263 (2000)
19. Ke, P., Sattler, U.: Next Steps for Description Logics of Minimal Knowledge and Negation as Failure. In: Baader, F., Lutz, C., Motik, B. (eds.) Proc. of Description Logics. CEUR Workshop Proceedings, vol. 353. CEUR-WS.org, Dresden, Germany (May 2008)
20. Krisnadhi, A.A., Sengupta, K., Hitzler, P.: Local closed world semantics: Keep it simple, stupid! In: Proc. of Description Logics (DL 2011). CEUR Workshop Proceedings, vol. 745.
21. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55(1), 1–60 (1992)
22. Motik, B., Rosati, R.: Reconciling Description Logics and rules. J. of the ACM 57(5) (2010)
23. Straccia, U.: Default inheritance reasoning in hybrid kl-one-style logics. In: Bajcsy, R. (ed.) Proc. of IJCAI 1993, pp. 676–681. Morgan Kaufmann, Chambéry, France (August 1993)

An algebraic characterization of unary two-way transducers [⋆] (Extended Abstract)

Christian Choffrut[1] and Bruno Guillon[1]

LIAFA, CNRS and Université Paris 7 Denis Diderot, France.

Abstract. Two-way transducers are ordinary finite two-way automata that are provided with a one-way write-only tape. They perform a word to word transformation. Unlike one-way transducers, no characterization of these objects as such exists so far except for the deterministic case. We study the other particular case where the input and output alphabets are both unary but when the transducer is not necessarily deterministic. This yields a family which extends properly the rational relations in a very natural manner. We show that deterministic two-way unary transducers are no more powerful than one-way transducers.

1 Introduction

In the theory of words, two different terms are more or less indifferently used to describe the same objects: transductions and binary relations. The former term distinguishes an input and an output, even when the input does not uniquely determine the output. In certain contexts it is a synonym for translation where one source and one target are understood. The latter term is meant to suggest pairs of words playing a symmetric role.

Transducers and two-tape automata are the devices that implement the transductions and relations respectively. The concept of multitape- and thus in particular two-tape automata was introduced by Rabin and Scott [7] and also by Elgot and Mezei [3] almost fifty years ago. Most closure and structural properties were published in the next couple of years. As an alternative to a definition via automata it was shown that these relations were exactly the rational subsets of the direct product of free monoids. On the other hand, transductions, which are a generalization of (possibly partial) functions, is a more suitable term when the intention is that the input preexists the output. The present work deals with two-way transducers which are such a model of machine using two tapes. An input tape is read-only and is scanned in both directions. An output tape is write-only, initially empty and is explored in one direction only. The first mention of two-way transducers is traditionally credited to Shepherdson [9].

Our purpose is to define a structural characterization of these relations in the same way that the relations defined by multi-tape automata are precisely

[⋆] Paper accepted at the *39th International Symposium on Mathematical Foundations of Computer Science*, Budapest, August 25-29, 2014, to appear.

the rational relations. However we limit our investigation to the case where the input and output are words over a one letter alphabet, i.e., to the case where they both belong to the free monoid a^* generated by the unique letter a. Our technique does not apply to non-unary alphabets. The input is written over one tape and is delimited by a left ($\triangleright$) and a right ($\triangleleft$) endmarker which prevents the reading head to fall off the input. An output is written on a second write-only tape. Formally a *two-way transducer* can be defined as a pair (A, ϕ) where A is a two-way automaton of transition set δ and ϕ is a production function, mapping δ into output words.

We now state our main result more precisely. Let Σ and Δ be respectively the *input* and the *output alphabets*. Given a binary relation $R \subseteq \Sigma^* \times \Delta^*$ and a word u in Σ^* we put $R(u) = \{v \mid (u, v) \in R\}$. We recall that R is rational if it belongs to the smallest family of subsets of $\Sigma^* \times \Delta^*$ which contains the finite languages and which is closed under set union, componentwise concatenation and Kleene star. We are able to prove the following

Theorem 1. *A relation of the monoid $a^* \times a^*$ is defined by a two-way transducer if and only if it is a finite union of relations R satisfying the following condition: there exist two rational relations $S, T \subseteq a^* \times a^*$ such that for all $x \in a^*$ we have*

$$R(x) = S(x)T(x)^*$$

The relation $\{(a^n, a^{kn}) \mid n, k \geq 0\}$ is a simple example. It is of the previous form, however it is not rational. Indeed, identifying a^* with the additive monoid of integers $\mathbb{N}$ this relation defines the relation "being a multiple of". However rational subsets of $\mathbb{N}$ are first-order definable in Presburger arithmetics, i.e., arithmetics with addition only.

We now briefly mention the few results which to the best of our knowledge are published on two-way transducers. Engelfriet and Hoogeboom showed links between two-way transducers and logic [4]. Filiot et al. have studied the simulation of functional two-way transducers by one-way transducer in [5].

2 Formal series

As suggested in introduction by $R(u)$ notation, a relation $R \subseteq \Sigma^* \times \Delta^*$ can be considered as a function from Σ^* into $\mathcal{P}(\Delta^*)$ or, equivalently, as a series over Σ^* with its coefficient in $\mathcal{P}(\Delta^*)$. The set of such series is denoted by $\mathcal{P}(\Delta^*)\langle\langle\Sigma^*\rangle\rangle$. This representation is the most convenient for our work. Thus, we will identify a relation R with its associated series $f_R : \ u \rightarrow \{v \mid (u, v) \in R\}$. In the same spirit we will speak of the *series accepted by a two-way transducer*. We use the traditional notation $\langle s, u\rangle$ in place of $f_R(u)$.

In addition to standard *rational operations* on series (sum, Cauchy product and Kleene star), we need two operations which we called *Hadamard-* or simply *H-operations*. The first one is the usual Hadamard product of two formal series: the coefficient of a word in a product is the product of the coefficients in the two series; the second happens to be new.

– the *Hadamard product* (or H-product): $s \circledH t : \forall u \in \Sigma^*,\ \langle s \circledH t, u \rangle = \langle s, u \rangle \langle t, u \rangle$
– the *Hadamard star* (or H-star): $s^{H\star} :\ \forall u \in \Sigma^*,\ \left\langle s^{H\star}, u \right\rangle = \langle s, u \rangle^*$

Denoting by Id the series associated to the identity relation, the series associated to the relation $\left\{(a^n, a^{kn}) \mid n, k \geq 0\right\}$ is equal to $Id^{H\star}$. The following is general but provides, when restricted to the case where Δ is unary, one direction of our main theorem 4.

Proposition 1. *If s and t are series accepted by two-way transducers, so are the series $s \circledH t$ and $s^{H\star}$.*

Rational series and beyond

The family of *rational series* over the semiring $\mathbb{K}$, denoted $\mathrm{Rat}_{\mathbb{K}} \langle\langle \Sigma^* \rangle\rangle$, is the smallest family of series over Σ^* with coefficients in the semiring $\mathbb{K}$ which contains the *polynomials*, *i.e.*, series with finitely many non empty coefficients, and which is closed under rational operations. The following result is classical [1, Theorem III. 7.1][2,8]:

Theorem 2. *The family of series in $\mathcal{P}(\Delta^*)\langle\langle \Sigma^* \rangle\rangle$ accepted by one-way transducers is equal to the family $Rat_{\Delta^*} \langle\langle \Sigma^* \rangle\rangle$.*

The family $\mathrm{Rat}_{\mathbb{K}} \langle\langle \Sigma^* \rangle\rangle$ is not closed under H-operations for an arbitrary semiring. However when $\mathbb{K}$ is commutative the following holds, [8, Thm III. 3.1]

Theorem 3. *If* $\mathbb{K}$ *is commutative then* $\mathrm{Rat}_{\mathbb{K}} \langle\langle \Sigma^* \rangle\rangle$ *is closed under H-product.*

The H-star of a rational series is not necessarily rational, even when Σ is unary. Therefore the following defines a broader family.

Definition 1. *The family of* Hadamard series*, denoted* $\mathrm{Had}_{\mathbb{K}} \langle\langle \Sigma^* \rangle\rangle$ *is the set of finite sums of Hadamard products of the form $\alpha \circledH \beta^{H\star}$ with $\alpha, \beta \in \mathrm{Rat}_{\mathbb{K}} \langle\langle \Sigma^* \rangle\rangle$.*

This family enjoys nice closure properties:

Proposition 2. *If* $\mathbb{K}$ *is commutative, the family* $\mathrm{Had}_{\mathbb{K}} \langle\langle \Sigma^* \rangle\rangle$ *is closed under finite sum, H-product and H-star.*

3 Unary two-way transductions

From now on we concentrate on unary two-way transducers, *i.e.*, on those with input and output alphabets reduced to the letter a and characterize the relations they define. We fix a transducer (A, ϕ).

The following is a reformulation of Theorem 1 in terms of series.

Theorem 4. *Let* $\mathbb{K}$ *denote the semiring* $\mathrm{Rat}(a^*)$. *A series $s \in \mathcal{P}(a^*) \langle\langle a^* \rangle\rangle$ is accepted by some two-way finite transducer if and only if $s \in \mathrm{Had}_{\mathbb{K}} \langle\langle a^* \rangle\rangle$, i.e., there exist a finite collection of rational series $\alpha_i, \beta_i \in \mathrm{Rat}_{\mathbb{K}} \langle\langle a^* \rangle\rangle$ such that:*

$$s = \sum_i \alpha_i \circledH \beta_i^{H\star}$$

The fact that the condition is sufficient is a direct consequence of Theorem 2 and Proposition 1. The other direction is more involved. We proceed as follows. We first show that if the transducer performs a unique *hit*, *i.e.*, it never visits endmarkers except at the beginning and at the end of the computation, it defines a rational relation. Then we use the closure properties of Property 2 to prove that the full binary relation with the possibility of performing an arbitrary number of hits, belongs to $\mathrm{Had}_{\mathbb{K}}\langle\langle a^*\rangle\rangle$.

We adapt a well-known construction based on *crossing sequences*, *i.e.*, sequence of destination states of transitions performed between two successive tape positions, in chronological order (see [6, page 36-42] for details). Using the commutativity of Δ^*, we are able to simulate by a one-way transducer, any *loop-free run* of (A, ϕ), *i.e.*, run that never visit the same position twice in the same state. Then, using again the commutativity of Δ^* and the fact that Σ is unary we extend this result to any hit, with or without loops. It is then possible to restrict this simulation to hits whose first and last configuration matches some fixed *border points*, *i.e.*, elements from $Q \times \{\triangleright, \triangleleft\}$ (the second component is the endmarker associated to the side of the tape).

Lemma 1. *Given a transducer, and two border points b_0 and b_1, there exists a computable one-way transducer that simulates any b_0 to b_1 hits.*

The accepted relation is thus rational, by Theorem 2.

Simulation of an unlimited number of hits

We first adapt the matrix multiplication to the Hadamard product. Let N be an integer and let be given two matrices $X, Y \in (\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)^{N\times N}$. We define the H-product of X and Y and also the H-star of X as the matrices:

$$X \circledH Y = \sum_{k=1}^{N} X_{i,k} \circledH Y_{k,j} \qquad (X)^{H\star} = \sum_{k=0}^{\infty} \overbrace{X \circledH \cdots \circledH X}^{k \text{ times}}$$

Proposition 3. *If the matrix X is in $(\mathrm{Had}_{\mathbb{K}}\langle\langle\Sigma^*\rangle\rangle)^{N\times N}$ then so is $(X)^{H\star}$.*

Now we are able to conclude the proof of Theorem 4.

Proof (Theorem 4). In one direction this is an immediate consequence of the fact that the family of series associated with a two-way transducer is closed under sum, Hadamard product and Hadamard star, see Proposition 1.

It remains to prove the converse. Let T be a transducer. Consider a matrix X whose rows and columns are indexed by the pairs $Q \times \{\triangleright, \triangleleft\}$ of border points. For all pairs of border points b_0 and b_1, its (b_0, b_1) entry is, by Lemma 1, the rational series associated to b_0 to b_1 hits of T. The series accepted by T is the sum of the entries of $X^{H\star}$ in position $((q_-, \triangleright), (q, \triangleleft))$ for q an accepting state. Since all rational series are also Hadamard series, we conclude by Proposition 2.

4 Conclusion

Our main result of Theorem 1 gives a characterization of relations (series) accepted by two-way unary transducers. A key point is that crossing sequences of loop-free runs have bounded size. In consequence, any loop-free runs can be simulated by a one-way transducer. We point out that this simulation does not require any hypothesis on the size of the input alphabet.

We fix a transducer $T = (A, \phi)$ accepting a relation $R \subseteq \Sigma^* \times \Gamma^*$, with $|\Gamma| = 1$. If A is deterministic or *unambiguous* (*i.e.*, for each input word u, there exists at most one accepting run of A on u), then every accepting run is loop-free. Therefore, by the previous remark, T is equivalent to some constructible one-way transducer. Another interesting case is when R is a function (T is *functional*). Then for each u, all the accepting runs on u produce the same output word. Hence, considering only loop-free runs preserves the acceptance of T.

Corollary 1. *Let $R \subseteq \Sigma \times \Delta$ with $|\Delta| = 1$ be accepted by some two-way transducer $T = (A, \phi)$. If A is unambiguous or if R is a function then R is rational.*

A *rational uniformization* of a relation $R \subseteq \Sigma^* \times \Gamma^*$, is a rational function $F \subseteq R$, such that the domain of F is equal to the one of R. Under the hypothesis $|\Gamma| = 1$, it is possible to build a one-way transducer accepting such a F. Since the transducer obtained from our work is not necessary functional, the construction involves a result of Eilenberg [2, Prop. IX 8. 2] solving the rational uniformization problem for rational relations.

Corollary 2. *There exists a computable one-way transducer accepting a rational uniformization of R.*

As a consequence of Lemma 1, in the case of unary transducers, the change of direction of the input head can be restricted to occur at the endmarkers only. In the literature such machines are known as *sweeping* machines [10].

References

1. J. Berstel. *Transductions and context-free languages.* B. G. Teubner, 1979.
2. S. Eilenberg. *Automata, Languages and Machines.* vol. A, Academic Press, 1974.
3. C. C. Elgot and J. E. Mezei. On Relations Defined by Finite Automata. *IBM Journal*, 10:47–68, 1965.
4. J. Engelfriet and H. J. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.
5. E. Filiot, O. Gauwin, P. A. Reynier, and F. Servais. From two-way to one-way finite state transducers. In *LICS*, pages 468–477, 2013.
6. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
7. M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.
8. J. Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.
9. J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development archive*, 3:198–200, 1959.
10. M. Sipser. Lower bounds on the size of sweeping automata. *J. Comput. Syst. Sci.*, 21(2):195–202, 1980.

Logspace computability and regressive machines

Stefano Mazzanti

Dipartimento di Culture del Progetto
Università Iuav di Venezia
Fondamenta delle Terese 2206, 30123 Venezia, Italy
email: mazzanti@iuav.it

Abstract. We consider the function class **E** generated by the constant functions, the projection functions, the predecessor function, the substitution operator, and the recursion on notation operator. Furthermore, we introduce regressive machines, i.e. register machines which have the division by 2 and the predecessor as basic operations. We show that **E** is the class of functions computable by regressive machines and that the sharply bounded functions of **E** coincide with the sharply bounded logspace computable functions.

Keywords: recursion on notation, logspace computable functions.

1 Introduction

One of the main features of logspace-algorithms is their incapacity to copy the whole input due to memory shortage, thus forcing them to read the data "on the fly", every time they are needed.

This feature has been exploited in [3] where the set **L** of logspace computable predicates has been shown to be the set of predicates recognized by read-only while programs and in [4] where the closure with respect to substitution and simultaneous recursion on notation of the constant functions and the projection functions has been shown to contain the characteristic functions of the predicates in **L**.

In this paper, we consider the class **E** of number theoretic functions defined as the closure with respect to substitution and (unbounded) recursion on notation of the predecessor function, the constant functions and the projection functions.

We show that **E** is a subset of the logspace computable functions which contains all the sharply bounded logspace functions. Moreover, we show that **E** is the set of functions computable by regressive machines, a kind of register machines which have the division by 2 and the predecessor as basic operations on registers.

Therefore, the present work can be considered an improvement of the characterization of **L** given in [4] because we use recursion on notation instead of simultaneous recursion and we characterize not only the $\{0,1\}$-valued logspace functions, but also the sharply bounded logspace computable functions.

2 Preliminaries

In this paper, we will only consider functions with finite arity on the set $\mathbb{N} = \{0, 1, \ldots\}$ of natural numbers.

From now on, we agree that x, y, z, i, j, n range over $\mathbb{N}$, that a, b, c, k range over $\mathbb{N} - \{0\}$, that $\mathbf{x}, \mathbf{y}, \mathbf{z}$ range over sequences (of fixed length) of natural numbers, that p, q range over integer polynomials with nonnegative coefficients and that f, g, h range over functions.

A function f is a *polynomial growth function* iff there is a polynomial p such that $|f(\mathbf{x})| \leq p(|\mathbf{x}|)$ for any $\mathbf{x}$, where $|x_1, \ldots, x_n| = |x_1|, \ldots, |x_n|$ and $|x| = \lceil \log_2(x+1) \rceil$ is the number of bits of the binary representation of x.[1] Moreover, f is *sharply bounded* iff there is a polynomial p such that $f(\mathbf{x}) \leq p(|\mathbf{x}|)$ for any $\mathbf{x}$, and f is *regressive* iff there is some constant k such that $f(\mathbf{x}) \leq \max(\mathbf{x}, k)$ for any $\mathbf{x}$, see [2].

We will use the following unary functions: the *binary successor* functions $s_0 : x \mapsto 2x$ and $s_1 : x \mapsto 2x + 1$; the *constant* functions $C_n : x \mapsto n$ for any $n \in \mathbb{N}$; the *division by two* function $div_2 : x \mapsto \lfloor x/2 \rfloor$; the *remainder* function $rem_2 : x \mapsto x - 2\lfloor x/2 \rfloor$; the *length* function $len : x \mapsto |x|$.

We will also use the following functions: the *modified subtraction* function $sub : x, y \mapsto x \dot{-} y = \max(x - y, 0)$; the predecessor function $P : x \mapsto x \dot{-} 1 = \max(x - 1, 0)$; the *bit* function $bit : x, y \mapsto rem_2(\lfloor x/2^y \rfloor)$; the *smash* function $smash : x, y \mapsto x \# y = 2^{|x| \cdot |y|}$; the *most significant part* function $MSP : x, y \mapsto \lfloor x/2^y \rfloor$. Recall that $bit(x, y)$ is the y-th bit of x and that $MSP(x, y)$ is the number represented in binary by the $|x| \dot{-} y$ leftmost bits of x. Finally, we will use the *substitution* operator $SUBST(g_1, \ldots, g_b, h)$ transforming functions $g_1, \ldots, g_b : \mathbb{N}^a \to \mathbb{N}$ and function $h : \mathbb{N}^b \to \mathbb{N}$ into the function $f : \mathbb{N}^a \to \mathbb{N}$ such that $f(\mathbf{x}) = h(g_1(\mathbf{x}), \ldots, g_b(\mathbf{x}))$ and the *recursion on notation* operator $RN(g, h_0, h_1)$ transforming function $g : \mathbb{N}^a \to \mathbb{N}$ and functions $h_0 : \mathbb{N}^{a+2} \to \mathbb{N}$ and $h_1 : \mathbb{N}^{a+2} \to \mathbb{N}$ into the function $f : \mathbb{N}^{a+1} \to \mathbb{N}$ such that

$$\begin{aligned} f(0, \mathbf{y}) &= g(\mathbf{y}) \,, \\ f(s_i(x), \mathbf{y}) &= h_i(x, \mathbf{y}, f(x, \mathbf{y})) \end{aligned}$$

where $i \in \{0, 1\}$. Let $\mathbf{E}$ be the closure under substitution and recursion on notation of the set comprehending the predecessor function P, the constant functions C_n for any n and the projection functions $I^a[i] : x_1, \ldots, x_a \mapsto x_i$ $(1 \leq i \leq a)$ with any arity a.

3 Regressive machines

Now, we introduce *regressive machines*, a kind of random access machines which have the division by 2 and the predecessor as basic operations.

[1] Here, the notation $|x_1, \ldots, x_n|$ is just an abbreviation for the sequence $|x_1|, \ldots, |x_n|$. However, $|x_1, \ldots, x_n|$ is also interpreted as $|x_1| + \ldots + |x_n|$.

A regressive machine operates on a finite number of variables (the registers) $\mathtt{X_1}, \ldots, \mathtt{X_b}$ and its program is built up according to the following rules:[2]

$$\mathtt{P} ::= \mathtt{X_i} := \mathtt{e} | \mathtt{pred(X_i)} | \mathtt{half(X_i)} | \mathtt{P_1;P_2} | \mathtt{loop\,X_i\,do\,P\,end}$$

where instruction $\mathtt{X_i} := \mathtt{e}$ sets the value of register $\mathtt{X_i}$ to the value of expression $\mathtt{e}$, and $\mathtt{e}$ can be any natural number constant, any register, or the least significant bit $\mathtt{lsb(X_j)}$ of register $\mathtt{X_j}$. Moreover, instructions $\mathtt{pred(X_i)}$ and $\mathtt{half(X_i)}$ compute the predecessor and (the quotient of) the division by 2 of $\mathtt{X_i}$, respectively. Finally, the program $\mathtt{loop\,X_i\,do\,P\,end}$ executes $|x|$ times program $\mathtt{P}$, where x is the value held by $\mathtt{X_i}$.

From now on, we adopt the notation of [4], so that registers and programs are denoted by capital letters and we will write programs with the typewriter font.

For any program $\mathtt{P}$ with b registers, a memory state of $\mathtt{P}$ is a sequence $\mathbf{x} = x_1, \ldots, x_b$ where x_i is the value of $\mathtt{X_i}$.

Consider the function $m_{\mathtt{P}} : \mathbb{N}^b \to \mathbb{N}^b$ such that $m_{\mathtt{P}}(\mathbf{x})$ is the state of $\mathtt{P}$ after the computation of $\mathtt{P}$ starting from state $\mathbf{x}$. The following lemma states that regressive machines compute regressive functions and that they operate in polynomial time as long as instructions are executed sequentially and each operation is executed in constant time.

Lemma 1. *For any program* $\mathtt{P}$ *with* b *registers there is a constant* c *such that* $I^b[i](m_{\mathtt{P}}(\mathbf{x})) \leq \max(\mathbf{x}, c)$ *for any* $1 \leq i \leq b$ *and there is a polynomial* p *such that the running time of* $\mathtt{P}$ *starting from memory state* $\mathbf{x}$ *is bounded by* $p(|\mathbf{x}|)$*.*

A program $\mathtt{P}$ with b registers computes a function $f : \mathbb{N}^a \to \mathbb{N}$ with respect to input registers $\mathtt{X_1}, \ldots, \mathtt{X_a}$ and output register $\mathtt{X_j}$ iff for any $x_1, \ldots, x_a$ the value $f(x_1, \ldots, x_a)$ is returned in register $\mathtt{X_j}$ when $\mathtt{P}$ is executed with $\mathtt{X_i}$ having initial value x_i for $1 \leq i \leq a$ and all the other variables are initialized to zero.

4 Main results

Let **SB** be the set of sharply bounded functions, let **RM** be the set of functions computable by regressive machines and let **FL** be the set of logspace computable functions. The following statement summarizes the relationships between logspace functions, class **E** and regressive machines.

Theorem 1.

$$\mathbf{FL} \cap \mathbf{SB} \subseteq \mathbf{E} \subseteq \mathbf{RM} \subseteq \mathbf{FL} \cap \mathbf{E}.$$

[2] The loop predecessor machines of [2] do not have either the assignment `X:=lsb(Y)` or the division instruction, and use the $\mathtt{LOOP\,X\,DO\,P}$ construct, which iterates x many times $\mathtt{P}$, where x is the value held by $\mathtt{X}$. In [3], counter machines with only decrement instructions have been considered, but the registers' contents are bounded by the length of the input.

Proof (Sketch). We first show that class $\mathbf{E}$ contains sharply bounded versions of the arithmetic operations (e.g. $add_p(x, y, z) = y + z$ for $y, z \leq p(|x|)$) and is closed with respect to the *sharply bounded maximization* operator $MAX_p(g)$ transforming function $g : \mathbb{N}^{a+1} \to \mathbb{N}$ into the function $f : \mathbb{N}^a \to \mathbb{N}$ such that $f(\mathbf{x}) = \max\{i \leq p(|\mathbf{x}|) | g(\mathbf{x}, i) \neq 0\}$ if $\{i \leq p(|\mathbf{x}|) | g(\mathbf{x}, i) \neq 0\} \neq \emptyset$, otherwise $f(\mathbf{x}) = 0$. Then, we set $bit_f(\mathbf{x}, i) = bit(f(\mathbf{x}), i)$ and show that $bit_f \in \mathbf{E}$ for any $f \in \mathbf{FL}$. The proof is carried out by induction on the characterization of $\mathbf{FL}$ given by Clote and Takeuti [1] which defines $\mathbf{FL}$ as the least function class closed with respect to substitution, concatenation recursion on notation and sharply bounded recursion on notation of the set comprehending the projection functions, the binary successor functions, the bit, the length and the smash functions.

Then, the first inclusion of the theorem is true because for any $g \in \mathbf{FL}$ such that $g(\mathbf{x}) \leq p(|\mathbf{x}|)$ for some polynomial p, we have

$$g(\mathbf{x}) = \max\{y \leq p(|\mathbf{x}|) | \forall_{i<|p(|\mathbf{x}|)|} bit(y, i) = bit_g(\mathbf{x}, i)\}$$

and the characteristic function of the predicate $\forall_{i<|p(|\mathbf{x}|)|} bit(y, i) = bit_g(\mathbf{x}, i)$ is in $\mathbf{E}$. The second inclusion can be easily obtained by showing (by induction on $\mathbf{E}$) that for any function $f \in \mathbf{E}$ there is a regressive machine computing f.

To show the third inclusion, we introduce counter machines and show that they simulate regressive machines using only a logarithmic amount of memory space. Then, counter machines can be easily simulated by functions in $\mathbf{FL} \cap \mathbf{E}$ and we obtain that $\mathbf{RM} \subseteq \mathbf{FL} \cap \mathbf{E}$. A *counter machine* operates on a finite number of read-only input registers and a finite number of read/write registers called *counters*. Input registers are denoted as $\mathtt{Y}_1, \ldots, \mathtt{Y}_\mathtt{a}$ and counters are denoted as $\mathtt{Z}_1, \ldots, \mathtt{Z}_\mathtt{b}$ for some a and b. Let $\mathbf{y} = y_1, \ldots, y_a$ be the input values and let $\mathbf{z} = z_1, \ldots, z_b$ be the values of the counters. A counter machine program is defined according to the following rules:

$$\mathtt{Q} ::= \mathtt{Z_i} := \mathtt{e} | \mathtt{succ(Z_i)} | \mathtt{half(Z_i)} | \mathtt{Q_1; Q_2} | \mathtt{if}\ (\mathtt{e_1} = \mathtt{n})\ \mathtt{then}\ \mathtt{Q_1}\ \mathtt{else}\ \mathtt{Q_2}$$
$$| \mathtt{loop}\ \mathtt{E_i}\ \mathtt{do}\ \mathtt{Q_1}\ \mathtt{end}$$

where $\mathtt{e}$ is any constant, any counter or $\mathtt{lsb(E_j)}$, expression $\mathtt{e_1}$ can be $\mathtt{Z_i}$, $\mathtt{bit(Y_{Z_i}, Z_j)}$ or $\mathtt{lsb(Z_i)}$, and $\mathtt{E_i}$ is an expression whose value is

$$e_i(\mathbf{y}, \mathbf{z}) = \begin{cases} z_{i+2} & \text{if } z_i = 0, \\ MSP(y_{z_i}, z_{i+1}) \dot{-} z_{i+2} & \text{otherwise} \end{cases} \quad (1 \leq i \leq b - 2).$$

Furthermore, we define the function $M_\mathtt{Q} : \mathbb{N}^{a+b} \to \mathbb{N}^a$ such that $(\mathbf{y}, M_\mathtt{Q}(\mathbf{y}, \mathbf{z}))$ is the memory state returned after the computation of a counter machine program $\mathtt{Q}$ starting from the state $(\mathbf{y}, \mathbf{z})$. Then, every regressive machine program $\mathtt{P}$ with b registers is *simulated* by a counter machine program $\mathtt{Q}$ with $3b$ registers. This means that for any $\mathbf{x} \in \mathbb{N}^b$, $\mathbf{y} \in \mathbb{N}^a$ and $\mathbf{z} \in \mathbb{N}^{3b}$, if $I^b[i](\mathbf{x}) = e_{3i-2}(\mathbf{y}, \mathbf{z})$ for any $1 \leq i \leq b$, then $I^b[i](m_\mathtt{P}(\mathbf{x})) = e_{3i-2}(\mathbf{y}, M_\mathtt{Q}(\mathbf{y}, \mathbf{z}))$ for any $1 \leq i \leq b$. In other words, the value of register $\mathtt{X_i}$ of program $\mathtt{P}$ is represented by counters $\mathtt{Z_{3i-2}}, \mathtt{Z_{3i-1}}, \mathtt{Z_{3i}}$ of program $\mathtt{Q}$ so that $e_{3i-2}(\mathbf{y}, \mathbf{z}) = x_i$. If $\mathtt{X_i}$ has been set to a constant value, then $z_{3i-2} = 0$ and z_{3i} is the value of $\mathtt{X_i}$. Otherwise, an input value has been assigned (or copied) to $\mathtt{X_i}$ and decrement or division instructions have been performed on it. In that case, the value of $\mathtt{X_i}$ is $MSP(y_{z_{3i-2}}, z_{3i-1}) \dot{-} z_{3i}$ (z_{3i-2} is the index of the input value, z_{3i-1} is the number of divisions and z_{3i} is less than or equal to the number of decrements). Then, $\mathtt{pred(X_i)}$ is simulated by $\mathtt{succ(Z_{3i})}$ (if $\mathtt{X_i}$ is positive) whereas $\mathtt{half(X_i)}$ is simulated by $\mathtt{succ(Z_{3i-1})}; \mathtt{half(Z_{3i})}$ ($\mathtt{Z_{3i}}$ could also be increased according to its parity and the value of $bit(y_{z_{3i-2}}, z_{3i-1})$).

So, for any function $f : \mathbb{N}^a \to \mathbb{N}$ computed by a regressive machine program $\mathtt{P}$ with b registers, there is a counter machine $\mathtt{Q}$ with $3b$ counters such that

$$f(\mathbf{x}) = e_{3j-2}(\mathbf{x}, M_{\mathtt{Q}}(\mathbf{x}, 1, 0, 0, \ldots, a, 0, 0, \ldots, 0))$$

where j is the index of the output register of $\mathtt{P}$. Moreover, by Lemma 1 there is a polynomial p such that $p(|\mathbf{x}|)$ bounds all the counters at any step of the computation of $\mathtt{Q}$. Therefore, we encode the counters with a single number $c_p(\mathbf{x}, \mathbf{z}) = z_1 p(|\mathbf{x}|)^{(3b-1)} + \ldots + z_{3b-1} p(|\mathbf{x}|) + z_{3b} < p(|\mathbf{x}|)^{3b}$ and we define functions $\tilde{e}_{p,i}, \tilde{M}_{p,\mathtt{Q}} : \mathbb{N}^{a+1} \to \mathbb{N}$ belonging to $\mathbf{FL} \cap \mathbf{E}$ such that $\tilde{e}_{p,i}(\mathbf{x}, c_p(\mathbf{x}, \mathbf{z})) = e_i(\mathbf{x}, \mathbf{z})$, $\tilde{M}_{p,\mathtt{Q}}(\mathbf{x}, c_p(\mathbf{x}, \mathbf{z})) = c_p(M_{\mathtt{Q}}(\mathbf{x}, \mathbf{z}))$ and

$$f(\mathbf{x}) = \tilde{e}_{p,3j-2}(\mathbf{x}, \tilde{M}_{p,\mathtt{Q}}(\mathbf{x}, c_p(\mathbf{x}, 1, 0, 0, \ldots, a, 0, 0, \ldots, 0))).$$

Since $c_p \in \mathbf{FL} \cap \mathbf{E}$, we obtain that $f \in \mathbf{FL} \cap \mathbf{E}$.

From Theorem 1 we obtain immediately that $\mathbf{E}$ is a subset of logspace computable functions and coincides with the class of functions computable by regressive machines.

Corollary 1. $\mathbf{E} = \mathbf{RM} \subseteq \mathbf{FL}$.

Moreover, by Theorem 1, we are also able to state that the sharply bounded logspace functions coincide with the sharply bounded functions in $\mathbf{E}$.

Corollary 2. $\mathbf{FL} \cap \mathbf{SB} = \mathbf{E} \cap \mathbf{SB}$.

Finally, from the corollary above we obtain the following new characterization of $\mathbf{L}$.

Corollary 3. *The characteristic functions of logspace predicates coincide with the* $\{0, 1\}$*-valued functions in* $\mathbf{E}$.

References

1. P. Clote and G. Takeuti, *First order bounded arithmetic and small boolean circuit complexity classes*, in *Feasible Mathematics II*, Birkhäuser Boston, 1995.
2. P. C. Fischer, J. C. Warkentin, *Predecessor Machines*, J. Comput. System Sci. 8 (1974) 190-219.
3. N. D. Jones, *LOGSPACE and PTIME characterized as programming languages*, Theoret. Comput. Sci. 228 (1999) 151-174.
4. L. Kristiansen, *Neat algebraic characterizations of LOGSPACE and LINSPACE*, Comput. Complexity 14 (2005) 72–88.

Author Index