Computational methods for Dynamic Answer Set Programming

Susana Hahn [1]

[1]University of Potsdam, Germany

Abstract

Dynamic problems, involving movement and change over time, arise in many real-world applications but remain difficult to handle in an integrated manner. This project addresses this challenge using Answer Set Programming (ASP) as a unified language for modeling, reasoning, and interacting with dynamic problems. We present contributions across three pillars: temporal extensions of ASP for representation, meta-programming as an implementation technique for reasoning, and ASP-driven user interfaces for interaction. We also outline future directions, from new dynamic formalisms to temporal interactivity and explanation.

1. Introduction

In our daily lives, we commonly encounter problems that require reasoning with time: determining our route to work, planning a trip, or scheduling our tasks. Such problems also arise in industry, in applications such as train scheduling and production sequencing, where their complexity increases significantly. These are commonly referred to as *dynamic problems*, as they involve movement and change over time. Building systems capable of handling dynamic problems is a non-trivial challenge. First, one must consider how a dynamic problem is *represented*. Several logical formalisms have been developed over the past decades for this purpose, allowing the use of concepts such as *after*, *eventually*, and *always*, enabling statements such as: *after the button is pushed, the light will eventually turn green*. Once the problem has been represented, various computational methods allow us to *reason* with it and obtain solutions, often through translations to other formalisms and structures. A further challenge arises when moving from theory to practice: implementing systems that combine the modeling power required to express dynamic formalisms with the computational capabilities needed to solve them. Moreover, for such a system to be usable by a broader audience, it must also provide a user interface that integrates with the underlying solver, supporting interactions such as the navigation of solutions and possible outcomes, the selection among possible actions at a given point in time, and the generation of explanations. Because each step in this process demands different expertise, the overall complexity of building a fully functional system for dynamic reasoning is substantial. As a result, most available systems tend to be single-purpose, tailored to a single logical formalism and provide domain-specific interfaces with limited functionality. This complexity makes it difficult for practitioners to explore different formalisms and techniques, as they are not usually able to test their ideas in practice and must rely on pen and paper to validate their theories. This project aims to address these challenges by providing a cohesive set of tools for modeling, reasoning, and interacting with dynamic problems.

Knowledge Representation and Reasoning (KRR) provides the theoretical foundation for this research. Interestingly, in this area there is a tight coupling between representation and reasoning: the same formalism that is used to express a problem often serves as the basis for computing its solutions. Among the various formalisms in this area, Answer Set Programming (ASP) stands out as a prime candidate due to its rich declarative modeling language and effective solvers. In particular, the ASP system *clingo* and its ecosystem of surrounding tools provide a powerful platform for modeling and solving complex problems. Although ASP offers strong foundations for knowledge representation and reasoning, it

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026) co-located with the Federated Logic Conference (FLoC 2026).

hahnmartinlu@uni-potsdam.de (S. H.)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-03

CEUR-WS.org/Vol-4311/DC_short_02.pdf

has no built-in support for temporal concepts. Nonetheless, its underlying logic has been extended in several directions to account for different temporal languages, and the *clingo* Python API has enabled the development of sophisticated extensions in practice. The resulting systems have been widely used, but their design and implementation is complex and not easily adaptable to other formalisms and techniques. This project aims to bridge this gap by providing ASP developers with the means to rapidly prototype dynamic systems end-to-end: from defining temporal extensions directly in ASP, to building intuitive interfaces for creating instances, visualizing solutions, and obtaining explanations — all within ASP, without requiring familiarity with other programming languages. In this way, the project targets researchers and practitioners who are fluent in ASP but face barriers when working across the broader software stack.

The path toward this goal has unfolded as a series of ASP extensions and systems. Each explores different logical foundations for enriching ASP's base logic with concepts from dynamic, temporal, and metric reasoning. For each extension, we have identified language fragments that provide the necessary modeling power while preserving properties that admit formalization and translation via established techniques. Implementing these extensions has led us to employ advanced meta-programming techniques in ASP, a process that has surfaced recurring difficulties and limitations. Drawing on this experience, we have developed *metasp*, a general-purpose framework for modeling ASP extensions via meta-programming. Although *metasp* is not specific to dynamic domains, it was designed with them in mind and has already been used to implement the extensions explored in this project. As part of this project, we have also developed *clinguin*, a framework for building ASP-driven user interfaces, and integrated it into *metasp* so that interfaces can be launched directly from the reasoning workflow. This integration is currently foundational, but serves as the basis for visualizing and interacting with computed plans. Together, these components now form a platform for exploring new extensions, reasoning techniques, and interactive features, serving both domain modelers and end users.

As next steps, we plan to investigate meta-programming techniques for incremental solving, the implementation of more expressive temporal languages, and a formal treatment of the distinction between actions and fluents. On the interaction side, we aim to develop the user interface into a monitoring system in which users can observe evolving solutions and interact through temporal constraints or sensory data. This interaction is envisioned to operate in the same dynamic language as the underlying extension, enabling users to pose queries and obtain explanations through explanation techniques adapted to dynamic domains. As a primary application domain, we plan to deploy these techniques on the FLATLAND benchmark, which formalizes the problem of train scheduling. We also intend to explore the application of Business Process Management, which would allow us to control and monitor the execution of arbitrary processes, including other *clingo* programs.

The remainder of this paper is organized around the three pillars of the project. Sections 2, 3, and 4 present the background, techniques, and current progress for representation, reasoning, and interaction with dynamic problems in ASP, respectively, followed by a discussion of future directions.

2. Representation

Dynamic logics One of the most commonly used temporal logics [1] is Linear-time Temporal Logic (LTL) [2]. LTL provides modal operators to express temporal properties such as $\circ$ (next), $\Box$ (always), and $\Diamond$ (eventually). LTL can also be defined in terms of Dynamic Logic (DL) [3], which allows for writing regular expressions over (infinite) traces and mixing declarative with procedural specifications. These types of specifications are also targeted by action languages such as GOLOG [4], which is based on situation calculus. Another interesting approach is Metric Temporal Logic (MTL) [5], which provides a means for measuring time differences between events by assigning a time value to states. MTL has been applied in domains such as scheduling [6], routing [7], and beyond [8]. A further extension of MTL is Metric Dynamic Logic (MDL) [9], which combines the features of DL and MTL, allowing for the expression of complex procedural knowledge with timing constraints. Originally, these formalisms were investigated for infinite traces. However, the finite-trace variant $\cdot_f$ has gained significant interest over

the past decade, as it aligns with a broad range of AI applications and admits more feasible computation. The introduction of LTL_f and Linear Dynamic Logic over finite traces (LDL_f) [10] served as a stepping stone for defining syntax and semantics under this restriction. Together, these formalisms support tasks such as planning, verification, and reactive control, and provide the logical foundations for the temporal extensions explored in this project.

Answer Set Programming Answer Set Programming (ASP) [11] is a well-established declarative problem-solving paradigm in which problems are encoded as logic programs. The combination of a rich modeling language and highly effective solving engines makes ASP a very attractive choice. ASP semantics can be formalized using equilibrium models [12] of the logic of here-and-there (HT) [13], which supports non-monotonic reasoning, where conclusions can be withdrawn in light of new information. HT has also been extended to here-and-there with constraints (HT_c) [14], incorporating linear constraints into the logic.

Temporal Logic Programming Temporal Logic Programming (TLP) emerged in the 1980s [15, 16, 17, 18] and was later revised in the context of ASP, giving rise to Temporal ASP. The central idea is to extend the equilibrium models of HT to account for dynamic scenarios, capturing time as sequences of HT-interpretations and yielding an expressive non-monotonic modal logic. Research began with infinite traces, producing (Linear) Temporal Here-and-There (THT) [19] and (Linear) Dynamic Logic of Here-and-There (DHT) [20], along with their non-monotonic counterparts: Temporal Equilibrium Logic (TEL) [19] and Dynamic Equilibrium Logic (DEL) [21]. This approach enabled the definition of Temporal Logic Programs [22]. More recently, TEL was extended with metric temporal operators over intervals of natural numbers [23], yielding Metric Equilibrium Logic (MEL). These three formalisms, namely TEL, DEL, and MEL, form the representational backbone of this project.

3. Reasoning

Reasoning with these logical representations requires systems that can process them and compute solutions efficiently. The ASP system *clingo* [24] is known for its high-performing engines, composed of the grounder *gringo*, which instantiates variables in the original logic program, and the solver *clasp*, which computes the solutions. Beyond these core components, *clingo* provides various tools for extending its language and customizing the solving process via a sophisticated Python API [25]. Its theory language capabilities allow the definition of custom syntactic expressions, their integration with specialized engines, and their translation to standard ASP rules. These features have led to the creation of several hybrid ASP systems. In particular, *clingcon* [26] and *clingo*[DL] [27] extend *clingo* with linear constraints under the semantics of HT_c.

In the temporal domain, this technique was applied to extend *clingo* with TEL_f, enriching its modeling power and yielding the first temporal ASP solver *telingo* [28]. This system uses *clingo*'s theory extension capabilities together with multi-shot solving in an incremental manner. Subsequent work incorporated dynamic operators from DEL_f [21, 29] by unfolding their definitions into TEL_f, relying on the introduction of auxiliary atoms in a Tseitin-style [30]. However, the complexity of this approach and its deep integration with *clingo*'s Python API make the resulting systems hard to maintain and extend.

Meta-programming An alternative technique that addresses these limitations is meta-programming, where the reification feature of *clingo* is used to represent the original program as a set of facts. A meta-program then consists of an ASP program that takes these facts as input and defines the semantics of the original program. This technique was presented in [31], showcasing how ASP semantics can be expressed within ASP itself. Unlike the Python API approach, having the semantics defined as an ASP program makes them more succinct and easier to adapt, extend, and maintain.

Our experience employing meta-programming for dynamic extensions, including the work translating dynamic formulas into automata structures[32], has surfaced recurring challenges. Since the approach relies on reification, it is affected by the stable-model-based simplifications performed during grounding. Users must prevent these simplifications with additional directives, which is cumbersome and error-prone. The flexibility of the approach can also allow user mistakes to go undetected, such as typing errors in custom expressions. Additionally, we identified common patterns needed to extract relevant information from the reified program, such as the list of formulas and sub-formulas for a custom extension and the contexts in which they appear. These findings motivated the development of *metasp*, a system that aims to simplify the use of meta-programming for defining custom semantics and extensions in ASP. *metasp* provides means to define formal type specifications and nesting capabilities in ASP, using them to validate input and detect syntactic errors. It introduces a transformation pipeline that protects nested modalities from stable-model-based simplifications during grounding, and enriches the reification with additional information about formulas and sub-formulas. Using *metasp*, we have been able to define the semantics of TEL_f, DEL_f, and MEL_f concisely and effectively as meta-programs.

The modularity of *metasp* makes it possible to explore different translation approaches for the same extension, as illustrated by our work on MEL_f [33]. The translation is structured in two parts: the first handles state changes, following the same semantics as TEL_f, while the second accounts for the timed aspect of metric logic. For this second part, we explored two approaches: a translation targeting HT, and one targeting HT_c. The latter leverages the integration of *clingcon* within *metasp*, yielding a succinct and performant encoding of metric constraints as linear constraints.

As future work in reasoning, we plan to explore the implementation of additional formalisms such as MDEL[9], as well as new techniques including incremental solving for temporal extensions, which would allow the horizon to grow dynamically rather than being fixed at grounding time.

4. Interactivity

Usability is a crucial aspect of any system intended for practical use. Beyond documentation, examples, and informative error messages, allowing users to interact with a system is essential for reaching broader audiences and ensuring the system is used effectively. However, creating User Interfaces (UIs) typically requires expertise outside of KRR, and is often outsourced to other developers. This poses a limitation for practitioners who would benefit greatly from having a simple way to prototype interfaces during the ongoing development of a custom extension.

The *clinguin* system [34] was created with this in mind, allowing practitioners to write UIs in ASP itself. As input, *clinguin* provides access to information representing the current state, such as which atoms are present in all models (inferred conclusions) and which are present in some models (possible selections), along with a set of interactive actions for user input, browsing solutions, and more. The system has been successfully used to build prototype interfaces for applications such as study regulations [35] and product configuration [36]. Within *metasp*, we have integrated a basic *clinguin* interface that allows users to interact with the dynamic extensions. Figure 1 shows screenshots of this interface for a simple TEL_f example, where a traffic light is controlled by a pedestrian button that eventually turns the light green after being pushed. The interface displays solutions separated by steps, allowing the user to browse plans, select atoms expected at a given step, and inspect the temporal formulas satisfied at each step. Temporal formulas are shown in blue at the state where they hold; opaque atoms and formulas appear in some solution but not in the currently selected one.

The interface we have developed so far is a first step, but interactive systems for temporal domains come with additional considerations. We envision evolving this interface into a monitoring system, where users can guide the system during planning and execution: browsing possible futures along a timeline, selecting which branch to follow, and injecting new information such as sensory data or exogenous events that may trigger replanning or comparison with the current execution. Taking inspiration from Interactive Planning [37], Execution Monitoring [38], and Explainable AI Planning (XAIP) [39, 40], we aim to provide users with the means to understand, evaluate, and modify plans

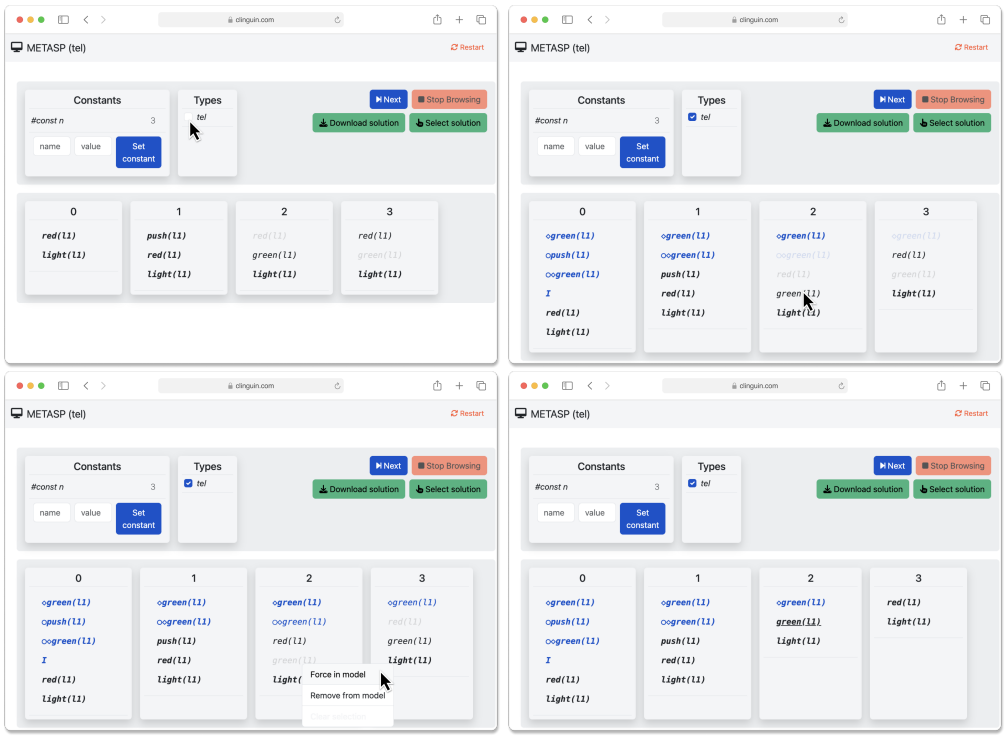


Figure 1: User interaction via mouse actions for the TEL_f extension in *metasp*.

within the same dynamic language used by the underlying extension.

A key component of such a system is the ability to explain the solutions produced. Temporal explanation is an active research area in its own, and its value is significantly amplified when integrated into an interactive system. We plan to explore contrastive temporal explanations [41], which allow users to ask why certain things are not happening in the current plan, and to receive answers in the form of minimal sets of actions and fluents that would need to change for a given temporal goal to be satisfied. Complementing this, temporal queries would give users a natural way to interrogate the system, separating questions that admit a yes/no answer from those that require a full contrastive explanation. Having a two-sorted language that distinguishes actions from fluents, would further support this by grounding explanations in the actions that drive temporal outcomes.

5. Conclusions

This PhD project aims to facilitate the development of dynamic ASP extensions across the full pipeline from representation to interaction. We have contributed to the formalization and implementation of several non-monotonic temporal formalisms, using these explorations to gather insights for the design of a general framework. The recently developed *metasp* system builds on these findings, providing means to define the syntax and semantics of ASP extensions within ASP itself and serving as a platform for exploring new techniques and formalisms. Its integration with *clinguin* lays the foundation for investigating temporal interactivity as the final pillar of this project.

Declaration on Generative AI

Generative AI was used exclusively for grammatical editing. All ideas and scientific content are the author's own.

References

[1] S. Demri, V. Goranko, M. Lange, Temporal Logics in Computer Science: Finite-State Systems, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

[2] J. Kamp, Tense Logic and the Theory of Linear Order, Ph.D. thesis, University of California at Los Angeles, 1968.

[3] D. Harel, J. Tiuryn, D. Kozen, Dynamic Logic, MIT Press, 2000. doi:10.1145/568438.568456.

[4] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. Scherl, GOLOG: A logic programming language for dynamic domains., Journal of Logic Programming 31 (1997) 59–83. doi:10.1016/S0743-1066(96)00121-5.

[5] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Systems 2 (1990) 255–299. doi:10.1007/BF01995674.

[6] R. Luo, R. Valenzano, Y. Li, C. Beck, S. McIlraith, Using metric temporal logic to specify scheduling problems, in: C. Baral, J. Delgrande, F. Wolter (Eds.), Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'16), AAAI Press, 2016, pp. 581–584.

[7] S. Karaman, E. Frazzoli, Vehicle routing problem with metric temporal logic specifications, in: 2008 47th IEEE conference on decision and control, IEEE, 2008, pp. 3953–3958. doi:10.1109/CDC.2008.4739366.

[8] U. Hustadt, A. Ozaki, C. Dixon, Theorem proving for pointwise metric temporal logic over the naturals via translations, Journal of Automated Reasoning 64 (2020) 1553–1610. doi:10.1007/s10817-020-09541-4.

[9] A. Becker, P. Cabalar, M. Diéguez, L. Fariñas del Cerro, T. Schaub, A. Schuhmann, Metric dynamic equilibrium logic, Journal of Applied Non-Classical Logics 33 (2023) 495–519. doi:10.1080/11663081.2023.2244365.

[10] G. De Giacomo, M. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: F. Rossi (Ed.), Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13), IJCAI/AAAI Press, 2013, pp. 854–860.

[11] V. Lifschitz, Answer Set Programming, Springer-Verlag, 2019. doi:10.1007/978-3-030-24658-7.

[12] D. Pearce, Equilibrium logic, Annals of Mathematics and Artificial Intelligence 47 (2006) 3–41. doi:10.1007/s10472-006-9028-z.

[13] A. Heyting, Die formalen Regeln der intuitionistischen Logik, in: Sitzungsberichte der Preussischen Akademie der Wissenschaften, Deutsche Akademie der Wissenschaften zu Berlin, 1930, pp. 42–56.

[14] P. Cabalar, R. Kaminski, M. Ostrowski, T. Schaub, An ASP semantics for default reasoning with constraints, in: R. Kambhampati (Ed.), Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI'16), IJCAI/AAAI Press, 2016, pp. 1015–1021. doi:10.5555/3060621.3060762.

[15] M. Fujita, S. Kono, H. Tanaka, T. Moto-Oka, Tokio: Logic programming language based on temporal logic and its compilation to Prolog, in: E. Shapiro (Ed.), Proceedings of the Third International Conference on Logic Programming (ICLP'86), volume 225 of *Lecture Notes in Computer Science*, Springer, 1986, pp. 695–709. doi:10.1007/3-540-16492-8_119.

[16] D. Gabbay, Modal and temporal logic programming, in: A. Galton (Ed.), Temporal Logics and their Applications, Academic Press, 1987, pp. 197–237.

[17] M. Abadi, Z. Manna, Temporal logic programming, Journal of Symbolic Computation 8 (1989) 277–295. doi:10.1016/S0747-7171(89)80070-7.

[18] M. Orgun, W. Wadge, Theory and practice of temporal logic programming, 1990, pp. 21–50.

[19] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, C. Vidal, Temporal equilibrium logic: a survey, Journal of Applied Non-Classical Logics 23 (2013) 2–24. doi:10.1080/11663081.2013.798985.

[20] A. Bosser, P. Cabalar, M. Diéguez, T. Schaub, Introducing temporal stable models for linear dynamic logic, in: M. Thielscher, F. Toni, F. Wolter (Eds.), Proceedings of the Sixteenth International

Conference on Principles of Knowledge Representation and Reasoning (KR'18), AAAI Press, 2018, pp. 12–21.

[21] P. Cabalar, M. Diéguez, T. Schaub, Towards dynamic answer set programming over finite traces, in: [42], 2019, pp. 148–162. doi:10.1007/978-3-030-20528-7_12.

[22] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, T. Schaub, A. Schuhmann, C. Vidal, Linear-time temporal answer set programming, Theory and Practice of Logic Programming 23 (2023) 2–56. doi:10.1017/S1471068421000557.

[23] P. Cabalar, M. Diéguez, T. Schaub, A. Schuhmann, Metric temporal answer set programming over timed traces, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Proceedings of the Sixteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'22), volume 13416 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2022, pp. 117–130. doi:10.1007/978-3-031-15707-3_10.

[24] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[25] R. Kaminski, T. Schaub, P. Wanko, A tutorial on hybrid answer set solving with clingo, in: G. Ianni, D. Lembo, L. Bertossi, W. Faber, B. Glimm, G. Gottlob, S. Staab (Eds.), Proceedings of the Thirteenth International Summer School of the Reasoning Web, volume 10370 of *Lecture Notes in Computer Science*, Springer-Verlag, 2017, pp. 167–203. doi:10.1007/978-3-319-61033-7_6.

[26] M. Banbara, B. Kaufmann, M. Ostrowski, T. Schaub, Clingcon: The next generation, Theory and Practice of Logic Programming 17 (2017) 408–461. doi:10.1017/S1471068417000138.

[27] T. Janhunen, R. Kaminski, M. Ostrowski, T. Schaub, S. Schellhorn, P. Wanko, Clingo goes linear constraints over reals and integers, Theory and Practice of Logic Programming 17 (2017) 872–888. doi:10.1017/S1471068417000242.

[28] P. Cabalar, R. Kaminski, P. Morkisch, T. Schaub, telingo = ASP + time, in: [42], 2019, pp. 256–269. doi:10.1007/978-3-030-20528-7_19.

[29] P. Cabalar, M. Diéguez, F. Laferriere, T. Schaub, Implementing dynamic answer set programming over finite traces, in: G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20), IOS Press, 2020, pp. 656–663. doi:10.3233/FAIA200151.

[30] G. Tseitin, On the complexity of derivation in the propositional calculus, Zapiski nauchnykh seminarov LOMI 8 (1968) 234–259.

[31] R. Kaminski, J. Romero, T. Schaub, P. Wanko, How to build your own asp-based system?!, Theory and Practice of Logic Programming 23 (2023) 299–361. doi:10.1017/S1471068421000508.

[32] P. Cabalar, M. Diéguez, S. Hahn, T. Schaub, Automata for dynamic answer set solving: Preliminary report, in: Proceedings of the Fourteenth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'21), 2021. URL: http://ceur-ws.org/Vol-2970/aspocpinvited1.pdf.

[33] A. Becker, P. Cabalar, M. Diéguez, S. Hahn, J. Romero, T. Schaub, Compiling metric temporal answer set programming, in: C. Dodaro, V. Martinez (Eds.), Proceedings of the Seventeenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'24), Lecture Notes in Artificial Intelligence, Springer-Verlag, 2024. To appear.

[34] A. Beiser, S. Hahn, T. Schaub, Asp-driven user-interaction with clinguin, in: P. Cabalar, T. Swift (Eds.), Technical Communications of the Fortieth International Conference on Logic Programming (ICLP'24), EPTCS, 2024.

[35] S. Hahn, C. Martens, A. Nemes, H. Otunuya, J. Romero, T. Schaub, S. Schellhorn, Reasoning about study regulations in answer set programming (preliminary report), in: J. Arias, S. Batsakis, W. Faber, G. Gupta, F. Pacenza, E. Papadakis, L. Robaldo, K. Rückschloß, E. Salazar, Z. Saribatur, I. Tachmazidis, F. Weitkämper, A. Wyner (Eds.), Proceedings of the International Conference on Logic Programming Workshops, volume 3437 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3437/paper5ASPOCP.pdf.

[36] J. Baumeister, S. Hahn, K. Herud, M. Ostrowski, J. Reutelshoefer, N. Rühling, T. Schaub, P. Wanko, Towards industrial-scale product configuration, Theory and Practice of Logic Programming (2026)

1–31. doi:10.1017/S1471068426100404.
[37] D. Wilkins, A. Robinson, An interactive planning system, Technical Report (1981).
[38] C. Fritz, Execution monitoring–a survey, University of Toronto, Tech. Rep (2005).
[39] M. Fox, D. Long, D. Magazzeni, Explainable planning, CoRR abs/1709.10256 (2017). URL: http://arxiv.org/abs/1709.10256.
[40] S. L. S. Vasileiou, W. Yeoh, T. C. Son, On the relationship between kr approaches for explainable planning, arXiv preprint arXiv:2011.09006 (2020).
[41] D. Kasenberg, R. Thielstrom, M. Scheutz, Generating explanations for temporal logic planner decisions, in: J. C. Beck, O. Buffet, J. Hoffmann, E. Karpas, S. Sohrabi (Eds.), Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020, AAAI Press, 2020, pp. 449–458. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/6740.
[42] M. Balduccini, Y. Lierler, S. Woltran (Eds.), Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19), volume 11481 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2019.