Delivering the Potential of Diagrammatic Logics

Gem Stapleton
University of Brighton
Email: g.e.stapleton@brighton.ac.uk

Abstract—**Diagrammatic notations and reasoning have become a prominent focus of research over the last two decades. We have now reached a point where the techniques required to formalize diagrammatic logics and prove meta-level results, such as soundness and completeness, are well understood. Moreover, we have insight into what makes effective diagrams. However, the majority of progress has been on diagrammatic logics that are very limited in expressiveness. Whilst such logics are exemplars of the current state-of-the-art and are useful in simple cases, they have not yet realized their full potential in real world applications. This paper summarizes the existing state-of-the-art in diagrammatic logics and poses a set of open questions. The paper will discuss the need for software tools to support the creation and use of diagrammatic logics which are needed for large-scale real-world take-up. Significant research is still necessary to deliver the full potential of diagrammatic logics.**

I. Introduction

Diagrammatic notations are widely used to convey information, reflecting their perceived benefits as a mode of communication. In mathematics, diagrams are often sketched as accompaniments to proofs or definitions, say, in order to illuminate their more formal presentation. However, the traditional presentation of formal or rigorous mathematics and, in particular, logic has used symbolic notations that are textual in style. In the case of logic, a mature branch of mathematics, the long held approach to formalization is to distinguish between the syntax and semantics. For classical logic, the semantics are typically defined using a model-theoretic approach. This then raises the question as to whether *diagrams* can be used as an equally formal alternative to symbolic logics. This was answered affirmatively by Shin, in seminal work during the 1990s [1], who not only formalized the syntax and semantics of the Venn-I and Venn-II logics, but also provided them with sound and complete inferences rules. Shin demonstrated that the syntax and semantics of diagrammatic notations can be defined just as rigorously as for symbolic logics.

Around the same time as Shin's seminal work, Hammer devised a sound and complete Euler diagram logic which had just three inference rules [2]. The last two decades have seen many more diagrammatic logics successfully developed. Euler diagrams, in particular, have been prominent in diagrammatic logics research, providing the basis for Swoboda and Allwein's Euler/Venn diagrams [3], Howse et al.'s spider diagrams [4], and Kent's constraint diagrams [5]. Moreover, Euler diagrams themselves have been investigated as a basis for syllogistic reasoning by Mineshima et al. [6]. Indeed, it has been shown, by Sato et al., that Euler diagrams lead to better understanding and ability to carry out inference tasks than symbolic approaches [7]. Thus, it is undeniable that Euler diagrams have formed a major component of research in this field. Other key examples of diagrammatic logics include Peirce's existential graphs [8], further developed by both Shin [9] and Dau [10].

Our knowledge about how to formalize diagrammatic logics has, since those early days of Shin's seminal contributions, considerably advanced. Typically, they are formally defined via an abstract syntax [11] and given a model theoretic semantics. Using an abstract syntax was found to overcome problems, identified by di Luzio [12], associated with attempting to reason about the logic at the concrete syntax level [13] (i.e., reasoning with the actual drawn diagrams). Formalized and *well-supported* diagrammatic logics with appropriate levels of expressiveness for real-world applications are well beyond the current state of the art. This is a substantial hinderance to realizing the significant potential of diagrammatic logics and needs to be addressed. In order to address this limitation, consideration needs to be given as to what scientific advances are necessary given how such diagrams might be applied. It is posited that the following are key areas that should be the focus of research, *developed in tandem rather than in isolation*, to deliver this potential:

1) **Expressiveness** Diagrammatic logics should be suitably expressive for intended application domains.
2) **Inference Systems** It should be possible to reason with the diagrammatic logic, to ensure that desirable properties follow from axioms defined, and that undesirable properties do not.
3) **Manual Diagram Drawing** To be practically applicable on a real-world scale, intelligent software must be provided that allows end-users to create and use diagrammatic statements.
4) **Automated Diagram Drawing** Software should be provided to automatically draw the results of inference rule applications, or translations from other notations.

All of the above need to have a strong emphasis on usability. If we are to realize a major goal of diagrams research (to provide accessible ways representing, and reasoning about, knowledge) then empirical evaluations are essential. The remainder of this paper is devoted to discussing these five aspects of diagrams research. Sections II to V correspond to the four areas listed above. Each of these sections briefly describes the existing state-of-the-art for the family of Euler diagram logics, highlights limitations and presents avenues for future work. Section VI concludes.

II. Expressiveness

Most of the existing diagrammatic logics have very limited expressiveness and are, therefore, not usable in a wealth of real-world applications. Of the Euler diagram family, most of them are monadic logics and cannot, therefore, talk about relationships between elements [1], [4], [14], [15]. Some extensions, such as Kent's constraint diagrams [5], formalized in [16], go beyond the monadic case, by using arrows to represent binary relations. Concept diagrams take the level of expressiveness beyond first-order, allowing quantification over sets, elements, and binary relations [17]. Table I summarizes the expressiveness of the family of Euler diagram-based logics, where: MFOL is monadic first-order logic, MFOL[=] is MFOL with equality, MFOL[≤] is MFOL with an order operator, DFOL[=] is dyadic FOL with equality, and DSOL[=] is dyadic second-order logic with equality .

TABLE I. THE EXPRESSIVENESS OF EULER DIAGRAM-BASED LOGICS

Diagrammatic Logic	Lower Bound	Upper Bound
Euler diagrams, as in [14]	MFOL	MFOL
Venn-II, as in [1]	MFOL	MFOL
Euler/Venn, as in [3]	MFOL	MFOL[=]
Spider diagrams, as in [15]	MFOL[=]	MFOL[=]
Spider diagrams with constants, as in [18]	MFOL[=]	MFOL[=]
Spider diagrams of order, as in [19]	MFOL[≤]	MFOL[≤]
Constraint diagrams, as in [16]	>MFOL[=]	DFOL[=]
Generalized constraint diagrams, as in [20]	DFOL[=]	<DSOL[=]
Concept diagrams, as in [17]	DSOL[=]	DSOL[=]

We now give a set of examples to illustrate differences between these levels of expressiveness. Consider the following sentences from the given symbolic logics:

(1) MFOL: $\forall x \neg(A(x) \wedge B(x)) \wedge \exists y A(y)$.
(2) MFOL[=]: $\forall x(A(x) \Leftrightarrow B(x)) \wedge \exists y \exists z(A(y) \wedge A(z) \wedge \neg(y=z))$.
(3) MFOL[≤]: $\forall x(A(x) \Leftrightarrow B(x)) \wedge \exists y \exists z(A(y) \wedge \neg A(z) \wedge y<z)$.
(4) DFOL[=]: $\forall x \forall y\big((A(x) \wedge R(x,y)) \Rightarrow B(y)\big)$.
(5) DSOL[=]: $\forall x \forall y \exists f\big((A(x) \wedge f(x,y)) \Rightarrow B(y)\big)$.

The first sentence can be expressed by all of the diagrammatic logics in table I. Examples of an Euler diagram and a Venn-II diagram expressing the same information are in Figs 1 and 2 respectively. The Euler diagram expresses $\forall x \neg(A(x) \wedge B(x))$ by using two non-overlapping closed curves (one for A and one for B). However, asserting $\exists y A(y)$ needs to be done indirectly, by turning the statement into $\neg \forall y \neg A(y)$. The statement $\forall y \neg A(y)$ is expressed by the righthand Euler diagram with the shading denoting that no elements can be in A. A horizontal bar, over the diagram, expresses negation. By contrast, the Venn-II diagram expressing (1) is more succinct, not requiring the use of any logical operators. This diagram expresses $\forall x \neg(A(x) \wedge B(x))$ by the use of shading. The $\otimes$-sequence asserts $\exists y A(y)$. In fact, the part of the $\otimes$-sequence inside both A and B is redundant, because we know no elements are inside both A and B, because of the shading.

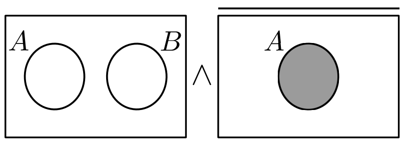


Fig. 1. Euler diagram for (1).

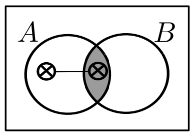


Fig. 2. Venn-II diagram for (1).

A spider diagram expressing (2) is in Fig. 3. It uses graphs (here, each graph comprises a single node) to represent the existence of elements, with distinct graphs representing distinct elements. Unlike spider diagrams, Euler/Venn diagrams do not include notation for explicitly representing the existence of elements. However, Euler/Venn diagrams *do* include notation to represent *specific* elements, i.e. constants, as do spider diagrams with constants. The Euler/Venn diagram in Fig. 4 expresses $\forall x \neg(A(x) \wedge B(x)) \wedge A(\text{tom}) \wedge B(\text{jerry})$. A spider diagram with constants expressing the same information is almost identical, shown in Fig. 5. Whilst it might appear that the inclusion of constants increases expressive power, this is actually not true. It can readily be shown that constants can be removed from logics, replacing them with existentially quantified formulae, without reducing expressiveness. See [18] for details in the case of spider diagrams with constants. Of note is that an extension of Shin's Venn logic includes constants, but its level of expressiveness is unknown [21], [22].

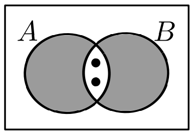


Fig. 3. Spider diagram for (2).

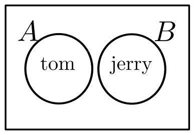


Fig. 4. An Euler/Venn diagram.

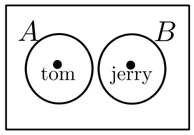


Fig. 5. A spider diagram with constants.

Statement (3) can be expressed by a spider diagram of order, introduced by Delaney [19], [23]. This variant of the spider diagram syntax expresses ordering information by augmenting the graphs (dots) with numbers [24], as well as a 'product' operator between diagrams. Numbers are placed on the nodes of graphs to indicate the relative ordering of the elements represented. The diagram corresponding to (3) is in Fig. 6, where the numbering, 1, of the dot inside A (and B) tells us that the represented element is ordered before the dot, numbered 2, outside A (and B). For this example, the product operator is not needed; see [19] for details and examples of its use.

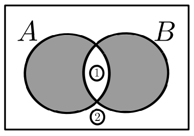


Fig. 6. Spider diagram of order for (3).

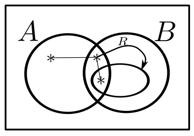


Fig. 7. Constraint diagram for (4).

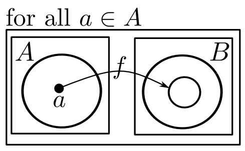


Fig. 8. Concept diagram for (5).

So far, all of the example diagrams given have been from monadic languages. They all use closed curves to represent sets (corresponding to 1-place predicates in MFOL) and, except for Euler diagrams, have explicit syntax to represent elements (sometimes unnamed elements, sometimes particular elements i.e. constants). Constraint diagrams and concept diagrams both build on this level of expressiveness, by including arrows to represent properties of binary relations. For example, the constraint diagram in Fig. 7 represents the same information as statement (4). The graph whose nodes are asterisks acts as a universal quantifier over A (as it is placed inside the curve labelled A). Thus, this graph can be thought of as representing all elements in A. The arrow, then, tells us that each of these elements is related only to elements in the set represented by the target of the arrow. In this example, the target is an unnamed subset of B. To summarize, every element in A is related to only elements in B under R.

Concept diagrams are similar to constraint diagrams except that they utilize quantifiers explicitly. The example in Fig. 8

represents statement (5). Here we see the use of a 'quantification expression', namely 'for all $a \in A$', written outside of the diagram's bounding box. Within this bounding box are two sub-diagrams. By placing the curves A and B inside different rectangles, concept diagrams avoid making assertions about the disjointness or subset relationships between the represented sets; see [25] for a discussion on how the use of multiple rectangles allows concept diagrams to reduce clutter and overcome over-specificity problems that commonly arise in diagrammatic notations.

Thus far, we have briefly detailed the expressiveness of a variety of diagrammatic logics based on Euler diagrams. There are various avenues of future work and here we pose three important open questions:

Q1: How can Euler diagram logics be extended to represent relations of arbitrary arity?
Q2: Can higher-order statements be effectively expressed by diagrammatic logics?
Q3: How effective are statements made in diagrammatic logics relative to those made in symbolic logics? Does any relative benefit decrease/increase as expressiveness increases?

Q3, in particular, represents a substantial programme of research and requires many empirical studies to be conducted. The results of such studies will be greatly illuminating and, potentially, help with the development of new diagrammatic logics. Q3 will also provide evidence (or otherwise) that serves to promote the use of diagrammatic approaches in place of their symbolic counterparts. Moreover, answers to these questions will aid with the application of diagrams to solving real-world problems. For instance, in the area of software modelling, for which constraint diagrams were proposed, there can be a need to make higher-order assertions, such as to define the transitive closer of a predicate or to quantify over sets.

III. Inference Systems

At the heart of any logic is its inference system. Indeed, a key aim of the diagrammatic reasoning community is to make proofs more accessible than those written using symbolic logics. For the purposes of this paper, a (formal) **proof** is a sequence of formulae where each formula is an axiom or derived, using an *inference rule*, from formulae written down earlier in the proof. Thus, in order to produce *diagrammatic* proofs, inference rules need to be developed for *diagrammatic* logics. Table II summarizes the state-of-the-art results for the development of inference systems for logics based on Euler diagrams. As the table indicates, the majority of the monadic logics are associated with a sound and complete inference system. Constraint diagrams, which we have seen include dyadic (2-place) predicates, whilst incomplete as a logic, does have sound and complete fragments such as [26]. The completeness proof strategies for *all* these logics rely on the decidability of the logics in question. Even though they are decidable, obtaining completeness is not always straightforward [27].

We now look, in more detail, at the inference rules that have been developed for a selection of these logics, starting with Venn-II. Shin's work on Venn-II (and Venn-I) [1] is widely regarded as the first formalization of a diagrammatic inference system. Both Venn-I and Venn-II are sound and complete. An example of an inference rule application, in Venn-II, can be seen in Fig. 9. The diagrams d_1 and d_2 are taken as axioms, with d_1 asserting that $A \neq \emptyset$ (or, in MFOL, $\exists x A(x)$) and d_2 asserting $A \cap C = \emptyset$ (equivalently, $\forall x \neg(A(x) \wedge C(x))$ in MFOL). From d_1 and d_2 we can deduce d_3, which expresses the same information but in a single diagram. Shin's so-called *unification* rule allows d_1 and d_2 to be combined into d_3. However, there is no single inference rule that allows d_4 in Fig. 10 to be deduced from d_1 and d_2 in just one step. This is perhaps surprising since d_4 is an obvious consequence of d_1 and d_2: d_4 merely takes d_2 and adds to it the information that $A \neq \emptyset$ given in d_1.

TABLE II. The soundness and completeness of Euler diagram-based logics

Diagrammatic Logic	Sound	Complete
Euler diagrams, as in [14]	Y	Y
Venn-II, as in [1]	Y	Y
Euler/Venn, as in [28]	Y	Y
Spider diagrams, as in [4]	Y	Y
Spider diagrams with constants, as in [29]	Y	Y
Spider diagrams of order, as in [19]	Y	N
Constraint diagrams, as in [30]	Y	N
Generalized constraint diagrams, as in [31]	Y	N
Concept diagrams, as in [25]	Y	N

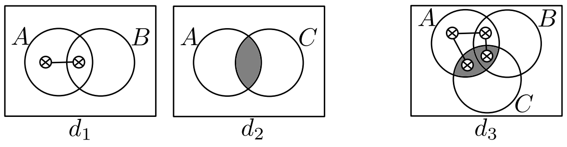


Fig. 9. Inference in Venn-II.

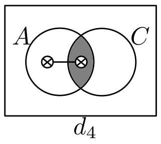


Fig. 10. Inference in Venn-II: simple deduction.

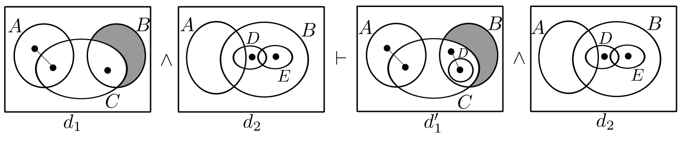


Fig. 11. A proof task using spider diagrams.

Examples of trivial deductions, like d_4, requiring non-trivial proofs are not unusual and certainly not confined to Venn-II. We now give a further, much more extreme, example of a trivial deduction requiring a non-trivial proof in the spider diagram logic. The proof task requires the deduction of $d'_1 \wedge d_2$ from the assumption $d_1 \wedge d_2$ shown in Fig. 11. We observe, from d_1 and d_2, the following information:

d_1: $B - C$ is empty (1), and
d_2: D is a subset of B (2).

Using (1) and (2), we can readily deduce D is a subset of $B \cap C$. The only difference between d_1 and d'_1 is the inclusion of the information that $D \subseteq B \cap C$. Intuitively, therefore, we see $d_1 \wedge d_2 \vDash d'_1 \wedge d_2$. The natural question then arises: how can we use spider diagram inference rules to prove $d_1 \wedge d_2 \vdash d_1' \wedge d_2$? A proof certainly exists because the logic is complete, but all proofs of $d_1' \wedge d_2$ are surprisingly long. One proof strategy is (loosely), to first add D to d_1 and then manipulate the syntax until we obtain d_1'. Using the inference rules in [4], the shortest proof that we have found takes, surprisingly, 24 steps.

The first step adds D, given in the first row of Fig. 12. The 'add contour' rule (the closed curves are called *contours* in spider diagrams) splits all regions, called *zones*, into two pieces. The graphs, which are called spiders, have twice the number of nodes after the rule has been applied. For each node in the original diagram, the two new nodes are placed in the two zones arising from the zone containing the original node. Semantically, this is because the element represented by the spider must lie either inside (the set denoted by) D or outside D. The proof must now use the information contained within d_3 and d_2 to 'move' D so that it is inside both B and C.

The next step is to apply a rule called *excluded middle* to d_3. This rule turns d_3 into a disjunction, with one new spider placed inside D, but outside B, to give d_4 and shading is added to the same region to give d_5. It is possible to show that d_4 and d_2 are in contradiction. Only one rule in the spider diagram logic allows contradictions to be identified and it requires that all spiders comprise single nodes, which is not the case for d_4. The proof will, later, identify this contradiction. Furthermore, we can also see that, in d_5, the element represented by the spider in A cannot also be in D, given the information in d_2. We can apply a rule called *splitting spiders* to d_5, giving d_6 and d_7 shown in the next line, turning this spider into two spiders, one inside $A-D$ (in d_6) and one inside $A \cap D$ (in d_7). As with d_4 and d_2, we now have d_7 and d_2 in contradiction. Again, the proof will eliminate this contradiction.

At this point in the proof, we now focus d_6. A spider diagram inference rule that allows the removal of shaded zones that contain no spiders can be applied, five times, removing the five such zones inside D. This moves D to inside both B and C, resulting in d_1'. Our diagram, at this stage in the proof, is now $(d_4 \vee d_1' \vee d_7) \wedge d_2$. To be able to identify that $d_4 \wedge d_2$ and $d_7 \wedge d_2$ are contradictions, we require all spiders to comprise just one node. We could now apply the splitting spiders rule to reduce the number of feet per spider, but this would result in an overly long proof. Instead, we remove information from d_4 and d_7 until only that needed for the contradiction to exist remains. For space reasons we omit the details, but the rest of the key steps in the proof are shown in Fig. 12. This 24 step proof is the shortest that we have been able to find that shows $d_1 \wedge d_2 \vdash d_1' \wedge d_2$. Clearly, for such an intuitively obvious result, this proof is not desirable.

One might ask why the proofs that arise using diagrammatic logics are not ideal given that a key ambition is to provide logics that are more accessible than their symbolic counterparts? The answer lies in the reason for which the inference rules were devised. Certainly in the case of spider diagrams, the inference rules were designed for obtaining a sound and complete system [4]. The nature of the inference rules for other diagrammatic logics, and their inherent usefulness for the completeness proofs given in the literature, suggests that the same holds for these other logics too. It is reasonable to conclude that the focus of inference rule development has been on obtaining *soundness* and *completeness*.

It is posited that the time is right for changing this focus, by designing inference rules that allow observable deductions to be made when writing proofs. Very recent work began, for spider diagrams, with this change of emphasis in mind [32]. New inference rules in [32] allow $d_1' \wedge d_2$ to be proved from $d_1 \wedge d_2$ in a single step, using the observable information about D in d_2 to add D to d_1. However, there is still a long way to go for this new approach to designing inference rule to result in improved logics, with the system in [32] only including five new rules all of which apply to diagrams of the form $d_1 \wedge d_2$.

Future challenges include answering the following questions:

- Q4: What constitutes a readable/understandable diagrammatic proof?
- Q5: How can inference rules for diagrammatic logics be designed so that they enable the production of readable/understandable proofs?
- Q6: Are the inference rules that allow the production of readable/understandable proofs also those that best allow proofs to be written by people?
- Q7: Is is possible to produce a sound and complete set of inference rules that allow readable/understandable diagrammatic proofs to be written?

As with the open problems given for expressiveness questions, these challenges will require contributions from cognitive science and need many empirical studies to be executed. As it stands, there is very little understanding about how to best design diagrammatic logics when aiming to make them effective tools for people to use. Defining such logics is important if they are to realize their full potential.

IV. Manual Diagram Drawing

If diagrammatic logics are to be useful in practice on a wide scale, such as in the area of ontology development [33], [34] then software tools are needed to support their use. A fundamental component of such tools is the ability of users to draw diagrams manually. There are numerous software tools that support diagram drawing, such as Inkscape or Word's diagram editing functionality. However, off-the-shelf tools do not offer sophisticated support for what can be complex tasks that must be performed using diagrammatic logics. These tasks include checking for consistency, debugging sets of axioms (i.e. determining whether the axioms define what was intended), and producing proofs using inference rules. Thus, dedicated tool support is needed.

We argue that such dedicated software for diagrammatic logics should support (at least) the following:

- (a) Sketch-based diagram drawing, using stylus-based input.
- (b) Diagram drawing using traditional point-and-click mouse based input.
- (c) Automatic understanding of the diagram syntax (both the concrete syntax and the abstract syntax).
- (d) Automated and interactive theorem proving support.
- (e) Automated diagram layout.

We now briefly discuss the first three requirements with respect to manual diagram drawing. The last two requirements will be discussed in the next section.

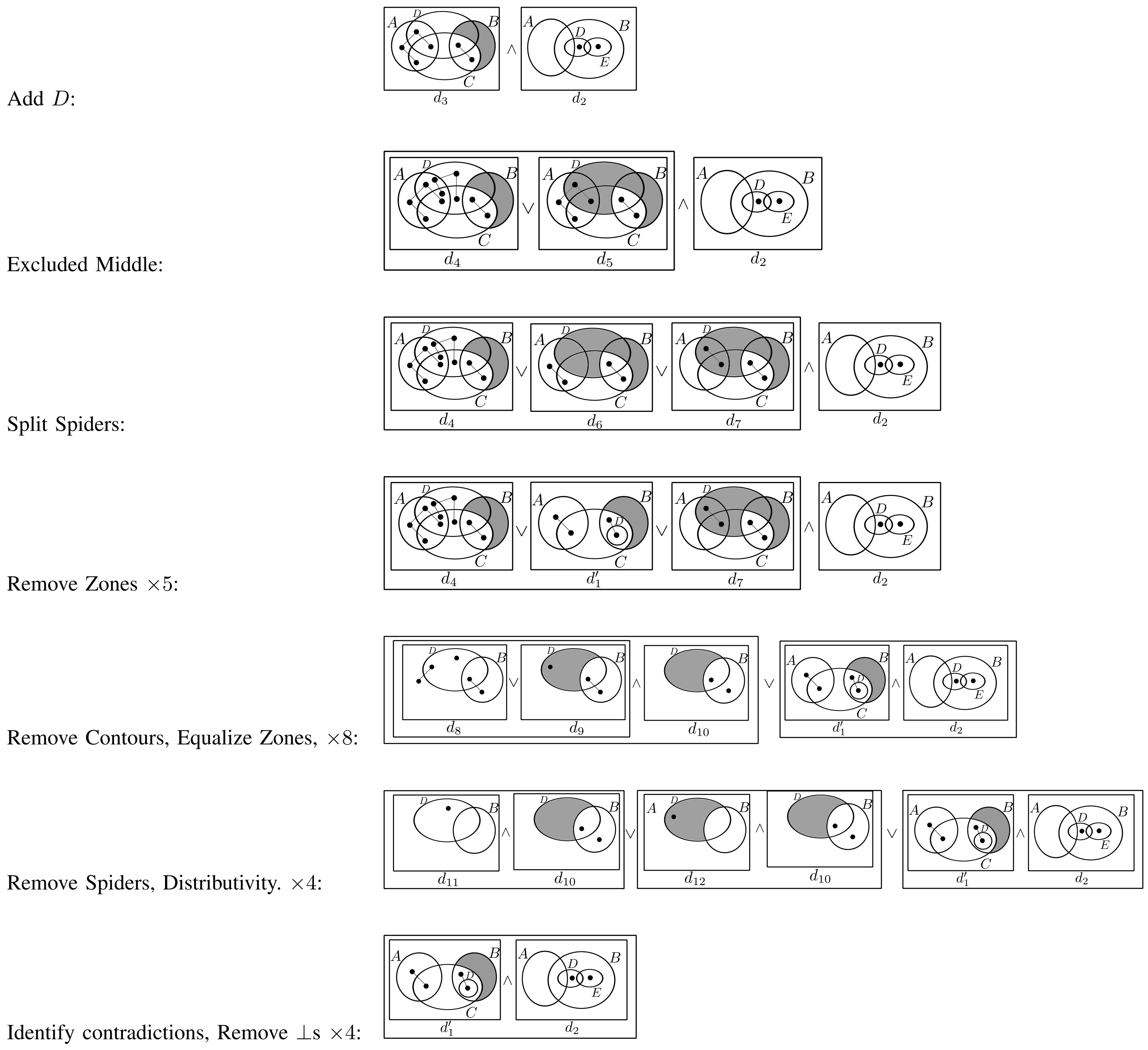
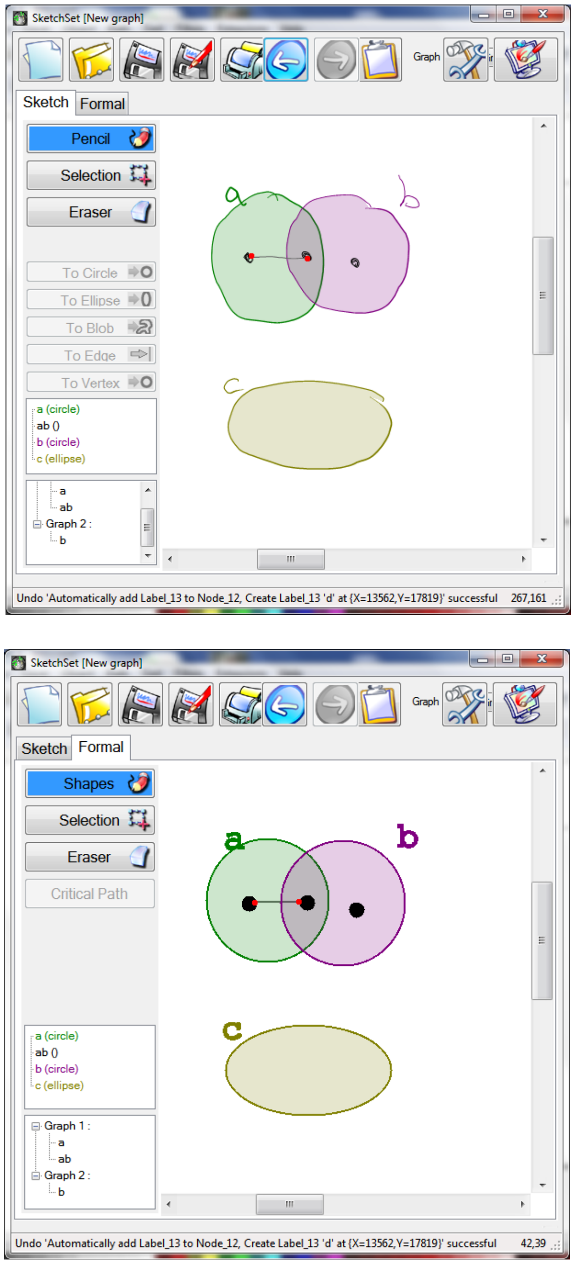


Fig. 12. A complex proof of a simple deduction.

Concerning (a), sketching tools have the advantage of providing natural interaction with the diagram and aid problem solving and communication [35]. Non-sketched diagrams (sometimes called *formal* diagrams, i.e. those created using traditional approaches) also have a role to play, in part because of the perception that sketches are incomplete, unfinished or inaccurate in some way [36]. In addition, the formal diagram is generally required for distribution. This supports our position that intelligent diagram creation systems should support visualization via formal diagrams, which appear as though they have been drawn in an editing tool rather than by hand, as well as sketched diagrams.

The provision of such tools requires sketch recognition technology to be developed for diagrammatic logics. Early work, by WAng et al., focused on Euler diagrams [37], and has been extended to include graphs (i.e. to spider-like diagrams) by Stapleton et al. [38]. An example can be seen in Fig. 13, which shows two screenshots of the SketchSet software [37]. The top image shows a manually drawn spider diagram which has been automatically converted to the formal diagram underneath. The interface also shows a stylized version of the abstract syntax (bottom left panels in each screenshot). The tool automatically computes the abstract syntax and uses it to ensure consistency between the sketch and formal diagrams. SketchSet allows edits to be made in each interface, automatically updating the other whilst maintaining consistency. Of note is that SketchSet utilizes a single-stroke recognizer to classify the sketched diagrammatic elements.



Undo 'Automatically add Label_13 to Node_12, Create Label_13 'd' at {X=13562,Y=17819}' successful 267,161

Undo 'Automatically add Label_13 to Node_12, Create Label_13 'd' at {X=13562,Y=17819}' successful 42,39

Fig. 13. Sketch recognition technology for spider diagrams.

Future challenges include answering the following questions:

Q8: Can sketch recognition technology be developed to allow multi-stroke recognition and stroke segmentation (when one stroke contributes to two or more syntactic elements) for diagrammatic logics?

Q9: What constitutes good user interface design for diagram creation tools that are specifically for diagrammatic logics?

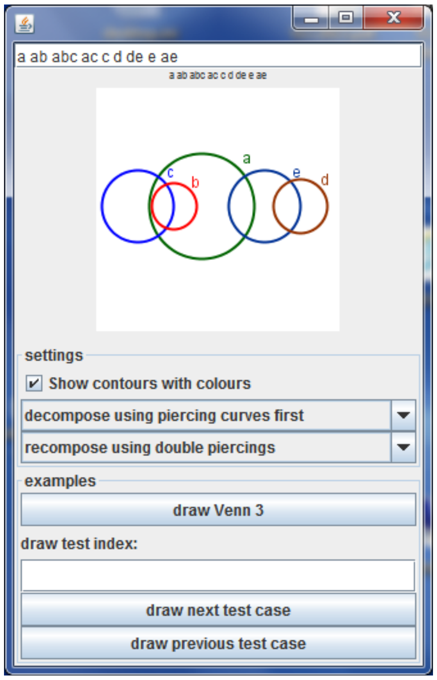
Q10: Can we produce computationally efficient algorithms for computing the abstract syntax of concrete diagrams, building on initial work for Euler and spider diagrams [37], [39], [40]?

The first of these challenges is a core problem faced by the sketch recognition community. Solving it will be important if diagram creation tools are to be able to handle the variety of ways in which people draw diagrams using a stylus.

V. Automated Diagram Drawing

In order to fully support interactive [41] and automated theorem proving [42], diagrams need to be automatically drawn on the application of an inference rule. For some inference rule applications, automatically drawing the resulting diagram is trivial, particularly if the inference rule merely deletes an item of syntax. However, some inference rules do not give rise to simple changes in diagram syntax. For instance, in Fig. 9, the diagram d_3 is not obtained from d_1 or d_2 by simple syntax deletion. Instead, it needs to be drawn using a more sophisticated approach. For this, algorithms are needed that automatically draw diagrams.

To-date, there has been considerable research effort towards automatically drawing Euler diagrams [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53]. These approaches start with the abstract syntax of the required diagram and proceed to seek a layout, often subject to some conditions (e.g. the curves in the resulting diagram must not self-intersect) [54]. Some of these automated diagram drawing (layout) methods use specific geometric shapes for the curves [45], [46], [50], [53]. An example of an automatically drawn Euler diagram, using circles, is in Fig. 14; a stylized form of the abstract syntax is shown at the top, entered by the user in order to create the diagram.



a ab abc ac c d de e ae

settings

[✓] Show contours with colours

decompose using piercing curves first

recompose using double piercings

Fig. 14. An automatically drawn Euler diagram [52].

Whilst recognizable geometric shapes are desirable, and circles are known to be both preferable [53] and most effective [55], not all diagrams can be drawn with them. Thus, other methods which allow arbitrary shaped curves, such as [44], [47] also have their place.

In order to produce the most effective diagram layouts, shape is not the only property that must be considered. To-date, empirical studies have been conducted exploring the impact of layout features on the comprehension of Euler diagrams. For instance, Blake et al. established that diagram orientation does not impact on user comprehension [56]. Related work by Benoy and Rodgers, showed that curves should be smooth and, when they intersect, they should diverge and zones should have roughly equal areas [57]. Other research has shown that well-formed Euler diagrams (see [54] for a list of well-formedness properties), better support comprehension than those which are not well-formed [58]. Some layout methods aim to produce well-formed diagrams only, such as the first ever method by Flower and Howse [44], extended by Rodgers et al. [48].

There are still a number of very challenging research problems to be solved in this area. Here we identify those we see as the most fundamental:

- Q11: How can we automatically draw diagrams that augment Euler diagrams with additional syntax?
- Q12: What are desirable/undesirable geometric and topological properties of logical diagrams, in terms of user comprehension and preference?
- Q13: Building on Q11 and Q12, how can we automatically draw diagrams for 'best' user comprehension?
- Q14: How can we automatically draw a set of diagrams that have common syntax?

For Q11, a naive approach is to automatically draw an Euler diagram and then add to it any additional syntax. However, as Fig 15 demonstrates, the best layout for the augmented diagrams need not be obtained in this way. On the left, the Euler diagram layout has compromised the addition of the graph whereas the layout on the right does not. Even partial solutions to Q12 will be able to inform layout choices, like that just illustrated, which will be key for answering Q13.

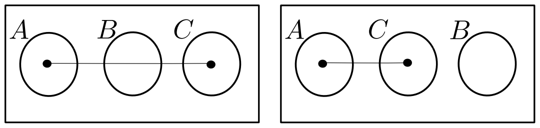


Fig. 15. Layout choices: impact on syntax.

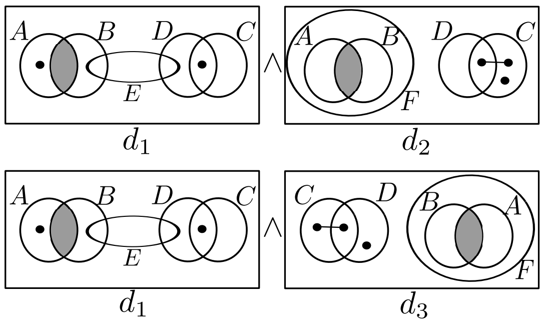


Fig. 16. Layout choices: in the context of logical connectives.

Q14 is particularly important for diagrammatic logics. For instance, a diagram that involves logical connectives could include diagrams with common parts, as in Fig. 16. Here, the common curves in d_1 and d_2 (namely A, B, C and D) adopt the same layout, rather than occupying significantly different positions as in $d_1 \wedge d_3$.

The only work, of which we are aware, that considers multiple Euler diagram layout (Q14) is by Rodgers et al. [59]. This work alters the layout of one diagram until it is similar to another. However, this approach is somewhat limited since it does not alter topological properties of diagrams. This means, for instance, that zone adjacency will never be altered. Thus, there are some pairs of diagrams that could both include Venn-4 as a sub-diagram that will never be made to look similar using the methods of [59]. More sophisticated approaches are needed for multiple diagram layout.

VI. Conclusion

This paper has summarized key results in diagrammatic logics research from a number of perspectives, focusing on expressiveness, inference, manual diagram drawing and automated diagram drawing. Whilst significant progress has been made, a number of open problems of some significance remain. We have presented some of these problems in this paper.

It is posited that solving these problems should not be done in isolation, but they should be informed by each other. Results achievable in one area will, no doubt, impact on the possible solutions in other areas. For instance, in the context of automated or interactive reasoning, the choice of inference rule application at each step could be guided by the automated layout algorithms that exist for the diagrammatic logic: the application of one inference rule may result in an ineffective layout for the resulting diagram, whereas another inference rule may yield an effective diagram layout. Optimizing proof readability and understandability will have to take into account results for diagram drawability.

References

[1] S.-J. Shin, *The Logical Status of Diagrams*. CUP 1994.

[2] E. Hammer, *Logic and Visual Information*. CSLI Publications, 1995.

[3] N. Swoboda and G. Allwein, "Using DAG transformations to verify Euler/Venn homogeneous and Euler/Venn FOL heterogeneous rules of inference," *Journal on Software and System Modeling*, vol. 3, no. 2, pp. 136–149, 2004.

[4] J. Howse, G. Stapleton, and J. Taylor., "Spider diagrams," *LMS Journal of Computation and Mathematics*, vol. 8, pp. 145–194, 2005.

[5] S. Kent, "Constraint diagrams: Visualizing invariants in object oriented models," in *Proceedings of OOPSLA97*. ACM, 1997, pp. 327–341.

[6] K. Mineshima, Y. Sato, R. Takemura, and M. Okada, "Towards explaining the cognitive efficacy of Euler diagrams in syllogistic reasoning: A relational perspective," *Journal of Visual Languages and Computing*, in press, 2013.

[7] Y. Sato, K. Mineshima, and R. Takemura, "The efficacy of Euler and Venn diagrams in deductive reasoning: Empirical findings," in *Diagrams*. Springer, 2010, pp. 6–22.

[8] C. Peirce., *Collected Papers*. Harvard University Press, 1933, vol. 4.

[9] S.-J. Shin, *The Iconic Logic of Peirce's Graphs*. Bradford Book, 2002.

[10] F. Dau, "Constants and functions in Peirce's existential graphs," in *Conceptual Structures*, 2007, pp. 429–442.

[11] M. Erwig, "Abstract syntax and semantics of visual languages," *Journal of Visual Languages and Computing*, vol. 9, no. 5, pp. 461–483, 1998.

[12] P. S. di Luzio, "Patching up a logic of Venn diagrams," *6th CSLI Workshop on Logic, Language and Computation*. CSLI, 2000.

[13] J. Howse, F. Molina, S.-J. Shin, and J. Taylor, "Type-syntax and token-syntax in diagrammatic systems," in *2nd International Conference on Formal Ontology in Information Systems*. ACM, 2001, pp. 174–185.

[14] G. Stapleton and J. Masthoff, "Incorporating negation into visual logics: A case study using Euler diagrams," in *Visual Languages and Computing 2007*. Knowledge Systems Institute, 2007, pp. 187–194.

[15] G. Stapleton, S. Thompson, J. Howse, and J. Taylor, "The expressiveness of spider diagrams," *Journal of Logic and Computation*, vol. 14, no. 6, pp. 857–880, 2004.

[16] A. Fish, J. Flower, and J. Howse, "The semantics of augmented constraint diagrams," *Journal of Visual Languages and Computing*, vol. 16, pp. 541–573, 2005.

[17] G. Stapleton, J. Howse, P. Chapman, A. Delaney, J. Burton, and I. Oliver, "Formalizing concept diagrams," in *19th International Conference on Distributed Multimedia Systems, Visual Languages and Computing*. Knowledge Systems Institute, 2013, pp. 182–187.

[18] G. Stapleton, J. Taylor, J. Howse, and S. Thompson, "The expressiveness of spider diagrams augmented with constants," *Journal of Visual Languages and Computing*, vol. 20, pp. 30–49, 2009.

[19] A. Delaney, "Defining star-free regular languages using diagrammatic logic," Ph.D. dissertation, University of Brighton, 2012, available at https://docs.google.com/open?id=0B18FG6I8GhB0b2hmT1lTc1hoeWM.

[20] G. Stapleton and A. Delaney, "Evaluating and generalizing constraint diagrams," *Journal of Visual Languages and Computing*, vol. 19, no. 4, pp. 499–521, 2008.

[21] L. Choudhury and M. K. Chakraborty, "On extending Venn diagrams by augmenting names of individuals," *Diagrams 2004*, LNAI 2980. Springer, 2004, pp. 142–146.

[22] L. Choudhury and M. Chakraborty, "On representing open universe," *Studies in Logic*, vol. 5, no. 1, pp. 96–112, 2012.

[23] A. Delaney, G. Stapleton, J. Taylor, and S. Thompson, "On the expressiveness of spider diagrams and commutative star-free regular languages," *Journal of Visual Languages and Computing*, vol. 24, no. 4, pp. 273–288, 2013.

[24] A. Delaney, J. Taylor, and S. Thompson, "Spider diagrams of order and a hierarchy of star-free regular languages," in *Diagrams 2008*, LNCS. Springer, 2008, pp. 172–187.

[25] P. Chapman, G. Stapleton, J. Howse, and I. Oliver, "Deriving sound inference rules for concept diagrams," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2011, pp. 87–94.

[26] G. Stapleton, J. Howse, and J. Taylor, "A decidable constraint diagram reasoning system," *Journal of Logic and Computation*, vol. 15, no. 6, pp. 975–1008, December 2005.

[27] J. Burton, G. Stapleton, and J. Howse, "Completeness proof strategies for Euler diagram logics," in *Euler Diagrams 2012*. CEUR, 2012, pp. 2–16.

[28] N. Swoboda and G. Allwein, "Heterogeneous reasoning with Euler/Venn diagrams containing named constants and FOL," *Euler Diagrams 2004*, ser. ENTCS, vol. 134. Elsevier Science, 2005.

[29] G. Stapleton, J. Howse, S. Thompson, J. Taylor, and P. Chapman, *Visual Reasoning with Diagrams*, ser. Studies in Universal Logic. Birkhauser, 2013, ch. On the Completeness of Spider Diagrams Augmented with Constants, pp. 101–133.

[30] A. Fish and J. Flower, "Investigating reasoning with constraint diagrams," in *Visual Language and Formal Methods 2004*, ser. ENTCS, vol. 127. Elsevier, 2005, pp. 53–69.

[31] J. Burton, G. Stapleton, and J. Howse, "Generalized constraint diagrams and the classical decision problem," *Journal of Logic and Computation*, vol. 23, pp. 199–262, 2013.

[32] G. Stapleton, M. Jamnik, and M. Urbas, "Designing inference rules for spider diagrams," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2013, pp. 19–26.

[33] F. Dau and P. Ekland, "A diagrammatic reasoning system for the description logic $\mathcal{ACL}$," *Journal of Visual Languages and Computing*, vol. 19, no. 5, pp. 539–573, 2008.

[34] J. Howse, G. Stapleton, K. Taylor, and P. Chapman, "Visualizing ontologies: A case study," in *International Semantic Web Conference*. Springer, 2011, pp. 257–272.

[35] G. Goldschmidt, *Visual and Spatial Reasoning in Design*. University of Sydney, 1999, ch. The Backtalk of Self-Generated Sketches, pp. 163–184.

[36] L. Yeung, B. Plimmer, B. Lobb, and D. Elliffe, "Effect of fidelity in diagram presentation," in *HCI 2008*. BCS, 2008, pp. 35–45.

[37] M. Wang, B. Plimmer, P. Schmieder, G. Stapleton, P. Rodgers, and A. Delaney, "Sketchset: Creating Euler diagrams using pen or mouse," in *IEEE Symposium on Visual Languages and Computing*. IEEE, 2011, pp. 75–82.

[38] G. Stapleton, A. Delaney, P. Rodgers, and B. Plimmer, "Recognising sketches of Euler diagrams augmented with graphs," in *Visual Languages and Computing*. KSI, 2011, pp. 279–284.

[39] R. Clarke, "Fast zone discrimination," in *Visual Languages and Logic*, 2007, pp. 41–54.

[40] G. Cordasco, R. D. Chiara, and A. Fish, "Efficient on-line algorithms for Euler diagram region computation," *Computational Geometry: Theory and Applications*, vol. 44, pp. 52–68, 2011.

[41] M. Urbas, M. Jamnik, G. Stapleton, and J. Flower, "Speedith: A diagrammatic reasoner for spider diagrams," in *Diagrams 2012*. Springer, 2012, pp. 163–177.

[42] G. Stapleton, J. Masthoff, J. Flower, A. Fish, and J. Southern, "Automated theorem proving in Euler diagrams systems," *Journal of Automated Reasoning*, vol. 39, pp. 431–470, 2007.

[43] S. Chow and F. Ruskey, "Drawing area-proportional Venn and Euler diagrams," in *Graph Drawing 2003*, LNCS 2912. Springer, 2003, pp. 466–477.

[44] J. Flower and J. Howse, "Generating Euler diagrams," in *Diagrams 2002*. Springer, 2002, pp. 61–75.

[45] H. Kestler, A. Muller, T. Gress, and M. Buchholz, "Generalized Venn diagrams: A new method for visualizing complex genetic set relations," *Bioinformatics*, vol. 21, no. 8, pp. 1592–1595, 2005.

[46] N. Riche and T. Dwyer, "Untangling Euler diagrams," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1090–1099, 2010.

[47] P. Rodgers, L. Zhang, G. Stapleton, and A. Fish, "Embedding well-formed Euler diagrams," in *12th International Conference on Information Visualization*. IEEE, 2008, pp. 585–593.

[48] P. Rodgers, L. Zhang, and A. Fish, "General Euler diagram generation," in *Diagrams 2008*. Springer, 2008, pp. 13–27.

[49] P. Simonetto, D. Auber, D. Archambault, "Fully automatic visualisation of overlapping sets," *Computer Graphics Forum*, vol. 28, no. 3, 2009.

[50] G. Stapleton, L. Zhang, J. Howse, and P. Rodgers, "Drawing Euler diagrams with circles: The theory of piercings," *IEEE Transactions on Visualisation and Computer Graphics*, vol. 17, no. 7, pp. 1020–1032, 2011.

[51] G. Stapleton, P. Rodgers, J. Howse, and L. Zhang, "Inductively generating Euler diagrams," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 1, pp. 88–100, 2011.

[52] G. Stapleton, J. Flower, P. Rodgers, and J. Howse, "Automatically drawing Euler diagrams with circles," *Journal of Visual Languages and Computing*, vol. 23, pp. 163–193, 2012.

[53] L. Wilkinson, "Exact and approximate area-proportional circular Venn and Euler diagrams," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 2, pp. 321–331, 2012.

[54] G. Stapleton, P. Rodgers, J. Howse, and J. Taylor, "Properties of Euler diagrams," *Layout of Software Engineering Diagrams*. EASST, 2007, pp. 2–16.

[55] A. Blake, G. Stapleton, P. Rodgers, L. Cheek, and J. Howse, "The impact of shape on the perception of Euler diagrams," in *under review*, 2014.

[56] ——, "Does the orientation of an Euler diagram affect user comprehension?" in *18th International Conference on Distributed Multimedia Systems*. Knowledge Systems Institute, 2012, pp. 185–190.

[57] F. Benoy and P. Rodgers, "Evaluating the comprehension of Euler diagrams," in *11th International Conference on Information Visualization*. IEEE, 2007, pp. 771–778.

[58] P. Rodgers, L. Zhang, H. Purchase, "Wellformedness properties in Euler diagrams: Which should be used?" *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 7, pp. 1089–1100, 2012.

[59] P. Rodgers, P. Mutton, and J. Flower, "Dynamic Euler diagram drawing," in *Visual Languages and Human Centric Computing, Rome, Italy*. IEEE Computer Society Press, September 2004, pp. 147–156.