Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 134 (2005) 127–151

www.elsevier.com/locate/entcs

A Survey of Reasoning Systems Based on Euler Diagrams

Gem Stapleton[1]

Visual Modelling Group
University of Brighton
Brighton BN2 4GJ, UK

Abstract

Euler diagrams have been used for centuries as a means for conveying ideas in an intuitive, informal way. Recently much research has been conducted to develop formal, diagrammatic reasoning systems based on Euler diagrams. Most of these systems extend Euler diagrams by adding further syntax to increase expressiveness. In this paper we survey such systems and draw comparisons between them.

Keywords: Visual Logic, Diagrammatic Reasoning.

1 Introduction

Euler diagrams [2] (sometimes called Euler circles) are a simple visual language for expressing logical statements. They exploit topological properties of enclosure, exclusion and intersection to represent subset, disjoint set and set intersection respectively. The diagram in figure 1 is an Euler diagram and expresses the fact that 'all tigers are cats and no dogs are cats' since (the contour labelled) *Tigers* is inside *Cats* and the contours *Dogs* and *Cats* contain regions with no points in common.

We will refer to a region that is a connected component of the plane as a *zone*[2]. For example, in figure 1, the set of points in the plane inside *Cats*

[1] Email: g.e.stapleton@brighton.ac.uk

[2] A zone can be identified by a containing set of contours and excluding set of contours that form a partition of the contour set.

doi:10.1016/j.entcs.2005.02.022

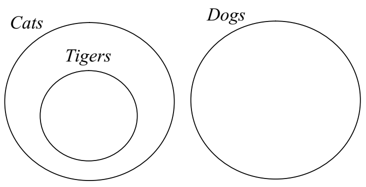

Fig. 1. An Euler diagram.

but outside $Tigers$ and $Dogs$ is a zone. Various semantics have been given for Euler diagrams. Zones in Euler diagrams represent sets and the union of all the sets represented by the zones in a diagram is the universal set. Some semantic interpretations specify that each zone in a diagram represents a non-empty set [31], whereas others do not impose this restriction [17]. A discussion of the semantics of Euler diagrams can be found in [19].

Euler diagrams are limited in expressiveness. For example, there is no Euler diagram that can express $A = \emptyset$ and nothing more. They can only express that a set is empty by using *missing zones* (a missing zone can be thought of as a zone which could be described by a partition of the contour label set but is not present in the diagram). Given certain well-formedness conditions on Euler diagrams (such as contours must be simple, closed, plane curves), there are less trivial examples of statements that Euler diagrams cannot express, identified in [30,52], because there is no drawable diagram with a specified zone set.

Many reasoning systems have emerged that extend Euler diagrams by using additional syntax to increase expressiveness. In this paper, we present a survey of such systems. In section 2, we outline Hammer's Euler diagram reasoning system [17]. We briefly discuss Venn diagrams in section 3. Peirce modified Venn diagrams, using x-sequences to assert the existence of elements [35]. We briefly outline his system in section 4. Shin's recent, seminal work on the Venn-I and Venn-II systems is discussed in sections 5 and 6 respectively [39]. Swoboda and Allwein develop a reasoning system based on Euler diagrams [45] that is similar to Shin's Venn-II system and we outline their work in section 7. Spider diagrams again extend Shin's Venn-II system [23]. Many sound and complete spider diagram systems have been developed, and we describe these in section 8. Constraint diagrams extend spider diagrams by using additional syntax [29]. They vastly increase expressiveness over spider diagrams and can express statements involving two place predicates. No other system outlined in this paper can make statements that require two place predicates (with the exception of equality). We include a discussion on constraint diagrams in section 9.

2 Reasoning with Euler Diagrams

A simple sound[3] and complete reasoning system based on Euler diagrams is given by Hammer in [17]. The system has just three reasoning rules: *the rule of erasure* (of a contour), *the rule of introduction of a new contour* and *the rule of weakening* (which introduces a zone).

Example 2.1 In figure 2 we illustrate Hammer's reasoning rules. Firstly we erase the contour labelled *Dogs* from the diagram d_1. This 'forgets' any information that we had about the set *Dogs*. Next we introduce the contour labelled *Mice*. When doing so, we must ensure that the new contour overlaps a proper part of each zone in the diagram, thus ensuring that we do not change the meaning of the diagram. Finally, we use the rule of weakening to introduce a new zone, giving d_2. By introducing the zone that is inside *Tigers* but outside *Cats* and *Mice*, we have 'forgotten' that all tigers are cats.

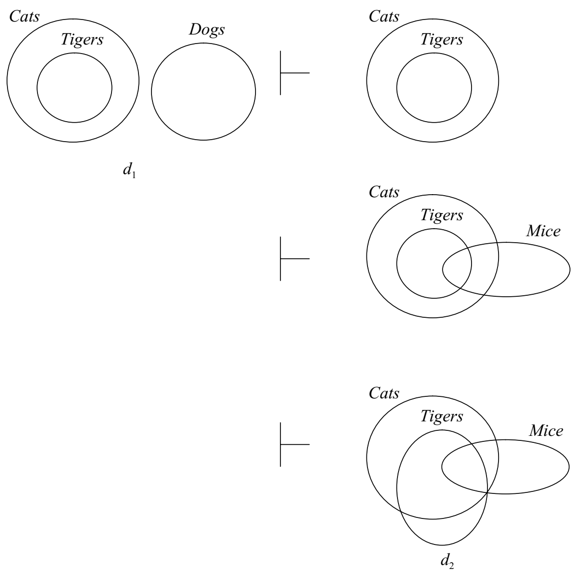

Fig. 2. Reasoning in Hammer's system.

Hammer's Euler diagram system is decidable. This follows as a corollary to the completeness proof strategy, which we now outline. Given $d_1 \vDash d_2$, Hammer constructs a proof from d_1 to d_2 (here, by proof we mean a sequence of reasoning rules applied to d_1 giving d_2). The first part of this proof is to add contours to d_1 until every contour label that occurs in d_2 occurs in d_1. Next, erase contours from d_1 until it has the same contour label set as d_2.

[3] Under the semantics that allow zones to represent the empty set.

Finally use the rule of weakening to give d_2. It can be shown that if the rule of weakening cannot be applied at this final stage then $d_1 \not\models d_2$.

3 Venn diagrams

Venn diagrams [50] are similar to Euler diagrams. However, instead of using missing zones to express that a set is empty, shading is used. All possible intersections between contours occur in Venn diagrams. Thus, the language of Venn diagrams extends a fragment of the Euler diagram language by using shading to represent the empty set. The Venn diagram in figure 3 expresses

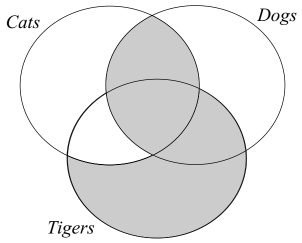

Fig. 3. A Venn diagram.

the same information as the Euler diagram in figure 1. Venn proposes a constructive method for drawing any Venn diagram on n contours [51] and More gives a topological proof that the construction process in valid [33].

A survey of work on Venn diagrams can be found at [37].

4 Venn-Peirce diagrams

Venn diagrams cannot assert the existence of elements nor express disjunctive information. To overcome this, Peirce modified Venn diagrams by introducing symbols into the system to represent non-emptiness of a set as well as emptiness [35]. The symbols x and o are used to represent non-emptiness and emptiness respectively. Peirce also uses lines to connect x's and o's, to represent disjunctive information. Shading is not used in Venn-Peirce diagrams.

Example 4.1 The Venn-Peirce diagram in figure 4 expresses that

(i) there is at least one real number and

(ii) all natural numbers are real numbers.

Peirce introduced six reasoning rules for his diagrammatic system [35], for example *we may connect any character to any existing character*. That is, we may join an x or o to any existing x or o using a line.

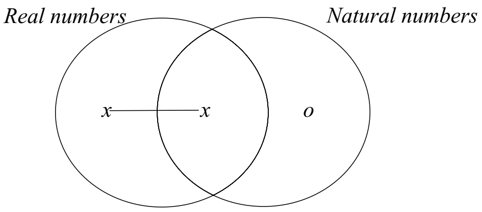

Fig. 4. A Venn-Peirce diagram.

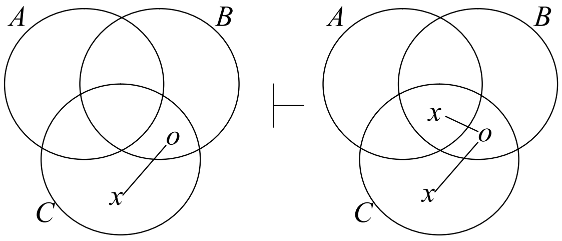

Fig. 5. Reasoning with Venn-Peirce diagrams.

Example 4.2 In figure 5, we can introduce an x, placed in the zone inside all three contours, and join it to the $x - o$ sequence. This rule is similar to the rule in propositional logic that allows $P \vee Q$ to be deduced from P.

A feature that distinguishes Venn-Peirce diagrams from the others discussed in this paper is the use of o instead of shading. Moreover, the fact that o can be joined to an x means that Venn-Peirce diagrams can express, for example, $A = \emptyset \vee B \neq \emptyset$ in a single diagram.

Peirce suggests an improvement to his notation: instead of using lines within a diagram to represent disjunctive information, draw diagrams containing only conjunctive information and use a collection of these diagrams to represent disjunctive information, like a type of disjunctive normal form. Later, we will review languages that allow single diagrams to be connected, using 'and' and 'or', increasing their expressiveness.

5 Venn-I diagrams

Shin [39] adapts Venn-Peirce diagrams by reverting back to Venn's shading to represent the emptiness of a set (rather than using o-sequences) and she draws a rectangle to represent the universal set. Because Shin uses shading instead of o-sequences, her Venn-I language is more restrictive and less expressive than the Venn-Peirce language. She also uses the symbol $\otimes$ in place of x.

Example 5.1 The Venn-I diagram d_1 in figure 6 expresses that

(i) no cats are dogs (by the use of shading),

(ii) all tigers are cats (by the use of shading),

(iii) there is at least one dog (by the use of an $\otimes$-sequence) and

(iv) there is at least one cat (by the use of an $\otimes$-sequence).

The diagram d_2 expresses the same as d_1: the extra $\otimes$-sequence inside *Dogs* but outside *Cats* and *Tigers* provides no extra information. So, $\otimes$-sequences simply assert non-emptiness of a set.

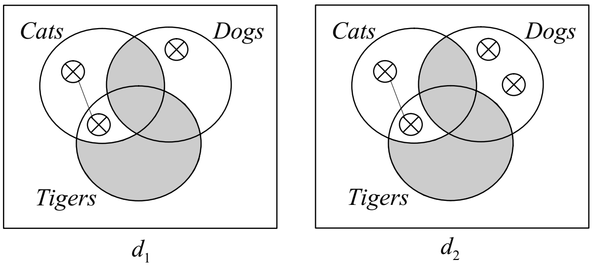

Fig. 6. Two Venn-I diagrams.

Shin defines six sound reasoning rules for Venn-I and proves completeness. One of the Venn-I reasoning rules allows the introduction of a contour. The contour must have a label that is not already present in the diagram and it must obey the *partial overlapping rule*: a new closed curve introduced into a given diagram should overlap a proper part of every zone in that diagram once and only once [39], page 57. Moreover, changes must be made to the $\otimes$-sequences, as illustrated in the following example.

Example 5.2 We can add a contour with a new label, C, to d_1, in figure 7, giving d_2. When introducing the contour, we must ensure that the shading is preserved. Moreover, we must ensure that all $\otimes$-sequences are modified so that each $\otimes$ is replaced by $\otimes - \otimes$, one part in each new zone.

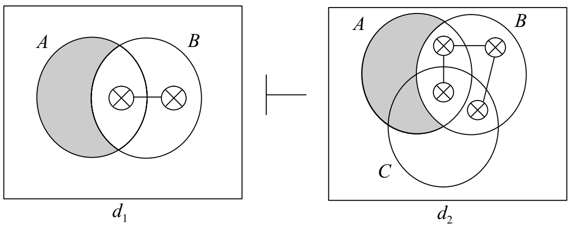

Fig. 7. Reasoning with Venn-I diagrams.

Another rule, called *the rule of unification of diagrams*, allows two diagrams to be replaced by a single diagram. The unify rule captures the conjunction of the semantic information in the two given diagrams (although the rule is defined purely syntactically) and replaces them by a single diagram, expressing this conjunctive information.

Example 5.3 We can unify d_1 and d_2, figure 8, giving d_3.

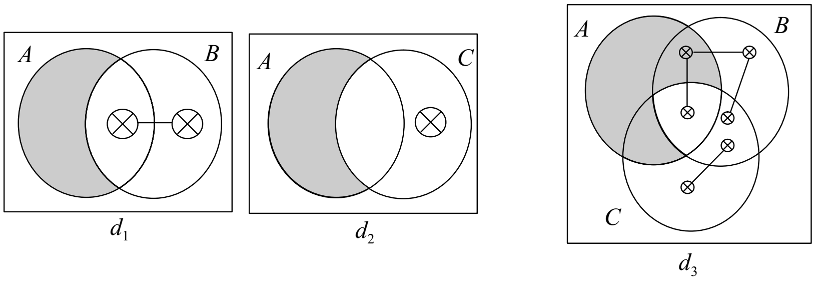

Fig. 8. Unifying Venn-I diagrams.

The remaining four rules required for completeness, given in [39] page 81, are described below.

(i) *The rule of erasure of a diagrammatic object.* Contours, a whole ⊗-sequence and the shading in an zone can be erased.

(ii) *The rule of erasure of part of an ⊗-sequence.* Part of an ⊗-sequence can be erased if that part is in a shaded region.

(iii) *The rule of spreading ⊗.* An ⊗ sequence can be introduced and connected to an existing ⊗-sequence.

(iv) *The rule of conflicting information.* If a shaded region contains an entire ⊗-sequence then the diagram can be replaced by any diagram. In this case, the ⊗-sequence asserts the non-emptiness of the set represented by the region and the shading asserts emptiness, which is a contradiction. From a contradiction, anything can be deduced.

6 Venn-II diagrams

Venn-I is limited in expressiveness and cannot express statements of the form $A \subseteq B \vee A \nsubseteq C$. Shin extends Venn-I to a more expressive system, called Venn-II, by allowing Venn-I diagrams to be connected by a straight line segments. Such a connecting line represents disjunction, like the lines connecting ⊗'s.

Example 6.1 The semantics of the Venn-II diagram in figure 9 is the disjunction of the semantics of its two Venn-I parts.

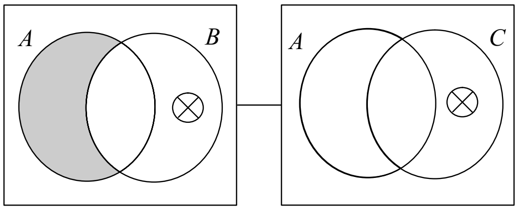

Fig. 9. A Venn-II diagram.

Shin defines ten reasoning rules for Venn-II and shows that they form a sound and complete set. The six rules for Venn-I generalize to Venn-II. The four new rules are as follows.

(i) *The rule of splitting ⊗-sequences.* An ⊗-sequence can be split into two (or, in general, more) pieces as follows. If D contains an ⊗-sequence, S, that is placed in two or more zones, then D can be replaced by $D_1 - D_2$ where each of D_1 and D_2 is a copy of D but neither contains S. The ⊗-sequence S is split into two pieces, one piece placed in D_1, the other in D_2.

(ii) *The rule of the excluded middle.* A diagram D that has an unshaded region, r, that does not contain an ⊗-sequence can be replace by a disjunction of diagrams, $D_1 - D_2$, where each of D_1 and D_2 is a copy of D but, in D_1, r is shaded and, in D_2, r contains an ⊗-sequence.

(iii) *The rule of connecting a diagram.* A diagrams D_1 can be replaced by $D_1 - D_2$ where D_2 is any diagram.

(iv) *The rule of construction.* $D_1 - ... - D_n$ can be replaced by D if each D_i can be replaced by D.

Venn-II diagrams cannot express arbitrary finite lower and upper bounds on the cardinalities of sets. Shin proves that Venn-II is equivalent in expressive power to pure monadic first order logic (in which there is no equality and all the predicate symbols are one place), which she calls L_0, and her proof strategy is algorithmic. To show Venn-II is at most as expressive as L_0, it is straightforward to translate any given diagram into a sentence. To find a diagram expressively equivalent to a sentence, she first converts the sentence into prenex normal form, say $Q_1x_1...Q_nx_nG$ where each Q_i is a quantifier and G is quantifier free. If Q_n is universal then G is transformed into conjunctive normal form. If Q_n is existential then G is transformed into disjunctive normal form. The quantifier Q_n is then distributed through G and as many formulae are removed from its scope as possible. All n quantifiers are distributed through the sentence in this way. The sentence resulting from this process has no nested quantifiers. A diagram can then be drawn for each of the simple parts of the resulting formula.

Example 6.2 Applying Shin's algorithm to the sentence $\exists x_1 \forall x_2 (P_1(x_1) \vee P_2(x_2))$ gives rise to the diagram shown in figure 10.

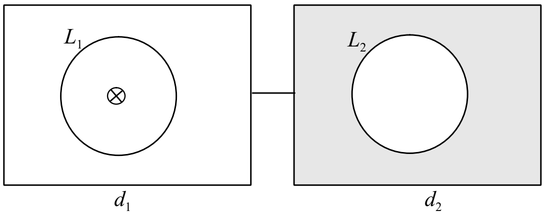

Fig. 10. Illustrating Shin's algorithm.

Very recently the Venn-II system has been extended to include constants [1].

7 Euler/Venn diagrams

Euler/Venn diagrams [45] are similar to Venn-I diagrams but, instead of $\otimes$-sequences, *constant sequences* are used. The interpretation of these symbols differs. An $\otimes$-sequence asserts non-emptiness of a set whereas a constant sequence asserts that a particular individual is in a set. Another difference is that Euler/Venn diagrams have underlying Euler diagrams whereas Venn-I diagrams are more restrictive, only allowing Venn diagrams as underlying diagrams. The diagram in figure 11 is an Euler/Venn diagram and expresses that no element is both a mammal and an insect and that there is something called 'tim' that is either a mammal or an insect.

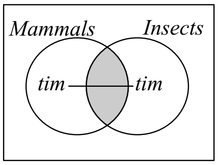

Fig. 11. An Euler/Venn diagram.

In [45], Swoboda gives a set of sound reasoning rules for Euler/Venn diagrams. These rules are extensions of those given by Shin and Hammer [17,18,39].

In [48] Swoboda and Allwein give an algorithm that determines whether a given Euler/Venn monadic first order formula is 'observable' [49] from a given diagram. Information is only observable from a diagram if it is explicitly represented in the diagram. If a formula is observable from a diagram then the formula is a consequence of the information contained in the diagram.

Example 7.1 The formula $P(bob) \vee Q(bob)$ is observable from the diagram in figure 12 but $P(chris) \vee \neg P(chris)$ is not, since $chris$ does not appear in the diagram.

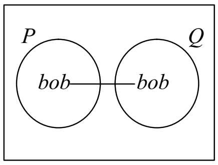

Fig. 12. Observable formulae.

The work in [49] on observation allows a recast relation (see [47]) to be defined between Euler/Venn diagrams and first order logic [46].

Two statements, S and T, (made in different languages) are related under a recast relation when the information expressed by S, say, can be extracted from T. In the particular case here, for example, an Euler/Venn diagram, D, is related to a first order logic sentence, S, under a recast relation when S is observable from D.

8 Spider diagrams

The work by Shin on Venn-I and Venn-II is seminal, and challenges the commonly held conception among mathematicians that diagrams cannot be used as formal tools, but only as an aid to understanding en-route to formal constructs such as proofs. Whilst the Venn-II system is not particularly expressive, Shin does show that diagrammatic notations can be formalized and given sound and complete reasoning rules.

Spider diagrams [13,22,25,26,32] adapt and extend Venn-II diagrams. Instead of using $\otimes$-sequences to indicate that regions represents non-empty sets, *spiders* are used to represent the existence of elements and, unlike $\otimes$-sequences, distinct spiders represent the existence of distinct elements. Thus spider diagrams allow finite lower bounds to be placed on the cardinalities of sets. In a Venn-II diagram, if shading is placed in the same region as an $\otimes$-sequence then the diagram is a contradiction. Furthermore, Venn-Peirce diagrams and Euler/Venn diagrams can also represent contradictions in a similar way. For spider diagrams, if shading is placed in the same region as a spider then the diagram expresses that all the elements in the set represented by that region are represented by its spiders. So shading, together with spiders, allows finite upper bounds to be placed on the cardinalities of the sets. Furthermore, like Euler/Venn diagrams, spider diagrams are based on Euler diagrams.

Example 8.1 The spider diagram d_1 in figure 13 expresses that no element is both a mammal and an insect and there are at least two elements in the set

$Mammals \cup Insects$. The spider diagram d_2 expresses that there are exactly two insects that are not mammals.

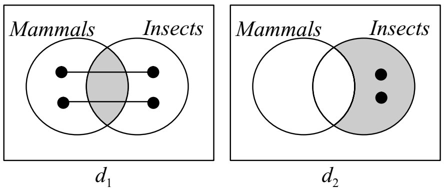

Fig. 13. Two spider diagrams.

In the example above, spiders represent the existence of elements and are called *existential spiders*. Some spider diagram systems instead use *given*, or *constant* spiders [26]. Constant spiders are always labelled and are analogous to constants in first order logic. We note here that the semantics of constant spiders and the semantics of constant sequences (used in Euler/Venn diagrams) are different: both represent particular individuals but, within a diagram, constant sequences with distinct labels do not necessarily denote distinct individuals, whereas constant spiders with distinct labels do denote distinct individuals.

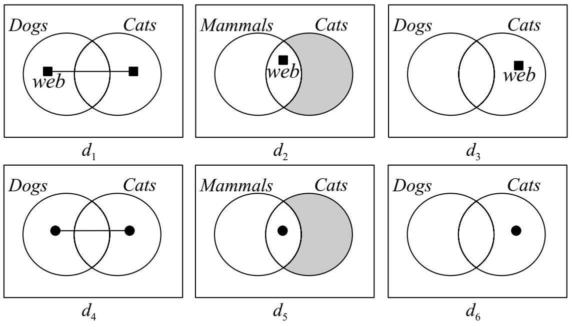

Fig. 14. Spider diagrams with constant spiders.

Example 8.2 The diagrams d_1, d_2 and d_3 in figure 14 all contain constant spiders (a constant spider is a labelled tree with square nodes). The diagram d_1 expresses that *web* is either a cat or a dog, but not both. The diagram d_2 expresses that *web* is a cat and a mammal and that all cats are mammals. From the conjunction of d_1 and d_2 we can deduce that *web* is a cat but not a dog, expressed by d_3. By contrast, from d_4 and d_5, which contain existential spiders (an existential spider is a tree with round nodes), we cannot deduce d_6. The choice of square nodes for constant spiders and round nodes for existential spiders is of no significance.

There are many spider diagram systems and in [26] a sound, but not complete, spider diagram system is presented that includes constant spiders but not existential spiders.

8.1 SD1 diagrams

The SD1 system was the first spider diagram system to be proved sound and complete [24,32]. SD1 does not include constant spiders but does include existential spiders. In SD1, existential spiders cannot have nodes placed in shaded zones. So, SD1 diagrams extend the Venn-II system by allowing lower bounds to be placed on the cardinality of sets. All diagrams in the SD1 system are based on Venn diagrams rather than Euler diagrams.

Example 8.3 The diagram in figure 15 is an SD1 diagram. It expresses that $C = \emptyset$, $|U - (A \cup B \cup C)| \geq 2$, $|A \cap B| \geq 1$ and $|A| \geq 2$, where U is the universal set.

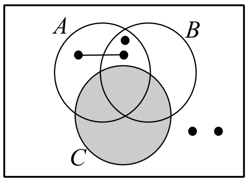

Fig. 15. An SD1 diagram.

Molina distinguishes between the *abstract syntax* and the *concrete syntax* [32], which are called the *type syntax* and *token syntax* respectively in [20,21]. The concrete syntax captures the physical representation of a diagram whereas the abstract syntax is a mathematical description of a concrete diagram. This has an analogy in graph theory. A graph is defined to be a set of vertices together with a set of edges, like the abstract syntax. An embedding of the graph in the plane is like the concrete syntax. Separating out these two levels of syntax overcomes the problems raised in [38] regarding the well-formedness of diagrams after applying a reasoning rule. Molina raises other issues that an abstract syntax overcomes, [32] pages 88-89:

> Shin introduces numerous notions, such as the set of all regions of a diagram, that seem to suggest the need of an abstract syntax to add precision to her diagrammatic system. ...
>
> The abstract syntax will give precision and will enable formality in different aspects of the notation. This will be perceived for instance with the formal descriptions of the reasoning rules where we can safely ignore certain aspects which are not needed for the intended meaning...
>
> Also a clear benefit of this distinction will be observed when building tools for SD1 or more expressive notations which may extend this system in the near future. For the design of the graphical user interface ... the concrete syntax will play a central role whereas the abstract syntax will be the guideline for the implementation of the symbolic calculus.

Swoboda and Allwein use Directed Acyclic Graphs (DAGs) as an abstract representation of Euler/Venn diagrams [48]. DAG transformations are used to check the correctness of reasoning steps. SD1 differs in that the reasoning system itself is defined at the abstract level, not the concrete level. The concrete level is used purely for visualizing abstract diagrams.

Example 8.4 The concrete SD1 diagram in figure 15 is called a *unitary* diagram and has the following abstract description.

(i) $C = \{c_1, c_2, c_3\}$ is a set of *contours*. The *boundary rectangle* is not an element of C.

(ii) $Z = \mathbb{P}C$ is the set of *zones*.

(iii) $Z^* = \{\{c_3\}, \{c_1, c_3\}, \{c_2, c_3\}, \{c_1, c_2, c_3\}\}$ is the set of *shaded zones*.

(iv) $R = \mathbb{P}Z - \{\emptyset\}$ is the set of *regions*.

(v) $R^* = \mathbb{P}Z^* - \{\emptyset\}$ is the set of *shaded regions*.

(vi) $L = \{A, B, C\}$ is the set of *contour labels*.

(vii) $S = \{s_1, s_2, s_3, s_4\}$ is a set of *spiders*.

(viii) $\eta \colon S \to \{r \in R : r \cap Z^*(d) = \emptyset\}$ is a function which returns the *habitat* of each spider defined by $\eta(s_1) = \{\emptyset\}$, $\eta(s_2) = \{\emptyset\}$, $\eta(s_3) = \{\{c_1, c_2\}\}$ and $\eta(s_4) = \{\{c_1\}, \{c_1, c_2\}\}$.

(ix) $l \colon C \to L$ is a function which returns the label of each contour defined by $l(c_1) = A$, $l(c_2) = B$ and $l(c_3) = C$.

With this abstract syntax, concrete unitary diagrams have non-unique abstractions. For example, in a concrete diagram with one contour, no spiders and no shading, any single element set, C, for the contours (together with the set of zones and so on) will suffice as an abstract description. Abstract unitary diagrams are defined on pages 63-64 in [32] and form the building blocks of *compound diagrams* and *multi-diagrams*. A compound diagram is a set of unitary diagrams and a multi-diagram is a set of compound diagrams.

Example 8.5 In figure 16, $\{d_1, d_2\}$ is a compound diagram and $\{\{d_1, d_2\}, \{d_3\}\}$ is a multi-diagram.

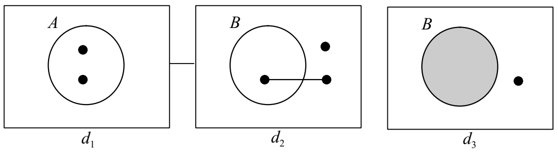

Fig. 16. A multi-diagram.

Semantically, distinct spiders represent the existence of distinct elements and shaded zones represent the empty set. The semantics of a compound diagram, D, are the disjunction of the semantics of the unitary diagrams in D. For a multi-diagram, Δ, the semantics are the conjunction of the semantics of the compound diagrams in Δ. So, a multi-diagram is in conjunctive normal form.

Molina also defines $\perp$ to be a false unitary diagram [32]. This allows unsatisfiable diagrams to be replaced by $\perp$ when reasoning rules are defined for the system. He defines twelve sound reasoning rules for SD1, similar to those for Venn-II, and proves this to be a complete set.

8.2 SD2 diagrams

In SD1, spiders are not allowed to have nodes placed in a shaded zones. So, SD1 diagrams can express finite lower bounds on the cardinalities of sets and that sets are empty but lacks facilities to express arbitrary finite upper bounds on the cardinalities of sets. The SD2 system addresses this issue, extending SD1 by allowing spiders to have nodes placed in shaded zones. In a shaded region, all of the elements are represented by spiders, allowing arbitrary finite upper bounds to be placed on the cardinalities of sets.

Example 8.6 The diagram in figure 17 is an SD2 diagram but not an SD1 diagram. The spider with two nodes represents the existence of an element which is either in B or $U - B$. In the first case, $|B| = 1$ and in the latter case, $|B| = 0$. The diagram express that $|U| \geq 2$, $|U - B| \geq 1$ and $|B| \leq 1$.

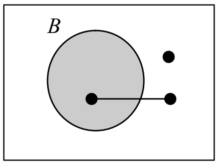

Fig. 17. An SD2 diagram.

We now outline two of the reasoning rules for SD2. Whilst the rules are specified at the abstract level, we will state the informal concrete level descriptions given by Molina. The abstract descriptions of the complete set of rules can be found in [32], pages 150-157.

Erasure of a spider. We may erase a spider from any completely non-shaded region.

Example 8.7 We can erase the spider, s, inhabiting $(A \cup B) - C$ in d_1, figure 18, and deduce d_2. The diagram d_1 asserts (amongst other things) that $|(A \cup B) - C| \geq 1$. Deleting s loses this cardinality information. Note that deleting the spider in $(A \cup C) - B$ is not a valid reasoning step: from d_1 we cannot deduce that $C = \emptyset$.

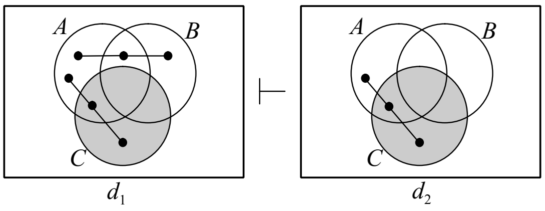

Fig. 18. Erasing a spider.

Splitting spiders. Let d be a unitary diagram with a spider s touching every zone of two disjoint regions, r_1 and r_2. Let d_1 and d_2 be two unitary diagrams that are copies of d except that neither contains s but each contains an extra spider whose habitat is r_1 in d_1 and r_2 in d_2. Then d can be replaced by the compound diagram $\{d_1, d_2\}$. The rule is reversible, that is, $\{d_1, d_2\}$ can be replaced by d.

Example 8.8 In figure 19, the spider in d asserts the existence of an element in either $A - C$ or $A \cap C$. We can split this spider and replace d by $\{d_1, d_2\}$.

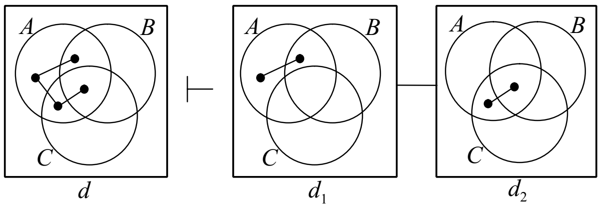

Fig. 19. Splitting spiders.

8.3 ESD2 diagrams

The ESD2 systems extends the SD2 system by allowing Euler based diagrams (rather than restricting to Venn diagrams) and using additional syntax [32].

Example 8.9 The diagram in figure 20 is an ESD2 diagram. This diagram contains a *tie*, which is a pair of parallel straight line segments, between the spider inside A and the spider inhabiting all the zones in the diagram. The tie allows us to express that if the two elements, x and y, represented by the spiders that are joined by the tie both satisfy $x, y \in A - B$ then those spiders represent the existence of the same element. The diagram also contains a *strand*, which is a wavy line segment, between the spider inside B and that inhabiting the entire diagram. The strand indicates that if the two elements, x

and y, represented by the spiders joined by the strand both satisfy $x, y \in B - A$ then they may represent the same element. The diagram expresses $A \cap B = \emptyset$ and

$$\exists x_1 \exists x_2 \exists x_3 \bullet x_2 \in A \wedge x_3 \in B \wedge (x_1 \in A \Rightarrow x_1 = x_2).$$

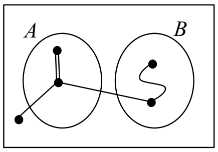

Fig. 20. An ESD2 diagram.

ESD2 is not more expressive than SD2. The reasoning rules for ESD2 are those for SD2 along with further rules required for completeness. The completeness proof for ESD2 is a 'bolt on' to the completeness proof for SD2: the first step is to prove every ESD2 diagram can be converted into a syntactically equivalent SD2 diagram, then the completeness of ESD2 follows from the completeness of SD2.

8.4 Further spider diagram systems

Whilst the ESD2 system allows Euler based diagrams, reasoning with ESD2 diagrams can produce unnecessarily long proofs: the need to first convert to SD2 form then reason and convert back to Euler form is not ideal. In [27] the SD2 system is modified to allow Euler based diagrams. We will refer to this extended system as SD3. All the reasoning rules for SD3 operate at the (abstract) Euler diagram level. Moreover, SD3 removes the restriction to conjunctive normal form (imposed in SD2). SD3 is the first reasoning system that we have reviewed which allows arbitrary connections using the 'and' and 'or' connectives. If d_1 and d_2 then so are $d_1 \wedge d_2$ and $d_1 \vee d_2$.

It has been shown that SD3 is equivalent in expressive power to monadic first order logic with equality ($MFOL_=$) [43,44]. The task of translating SD3 diagrams into logic sentences, thus showing that spider diagrams are at most as expressive as $MFOL_=$, is straightforward. In [44], it is shown that for every sentence, S, in $MFOL_=$ there exists a finite set of models that can be used to classify all the models for S. Each classifying model has a finite domain and can be used to construct a diagram. The disjunction of all such diagrams is expressively equivalent to S. The idea is illustrated in the following example.

Example 8.10 Let S be the sentence $\exists x A(x) \vee \forall x A(x)$. There are four classifying models for S that give rise to the diagrams d_1, d_2, d_3 and d_4 in figure 21.

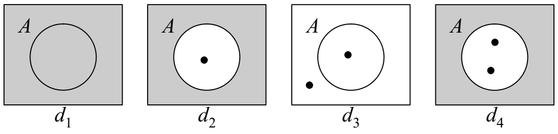

Fig. 21. Constructing diagrams from models.

The diagram $d_1 \vee d_2 \vee d_3 \vee d_4$ is expressively equivalent to S. This is not the 'natural' diagram one would associate with S.

Venn-II diagrams cannot express that a particular property holds for a unique element:

$$\exists x\,(A(x) \wedge \forall y\,(A(y) \Rightarrow x = y)).$$

Thus SD3 properly increases expressiveness over Venn-II. It has been shown that extending SD3 to include constant spiders does not lead to an increase in expressiveness [42].

Work is currently underway to extend SD3 to include *projected contours* [28]. Projected contours [15,16] allow us to represent partial relationships between sets.

Example 8.11 The diagram d_1 in figure 22 contains a projected contour labelled B. It expresses that A and C are disjoint and $A \cap B$ contains exactly one element. The diagram d_2 is semantically equivalent to d_1. The projected contour has reduced the number of zones.

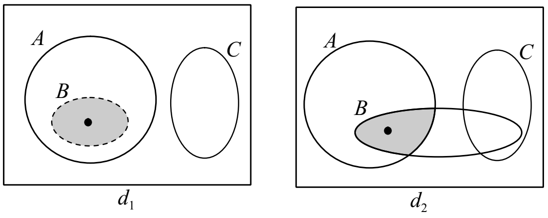

Fig. 22. Illustrating projected contours.

In a diagram without projections, to say nothing about the relationship between two sets requires the presence of zones, whereas with projections some zones need not be present.

Automated theorem provers have been developed for spider diagrams using both direct [11] and heuristic approaches [8,9]. The presentation of automatically generated proofs to users as sequences of concrete diagrams relies on the automatic generation of concrete diagrams from abstract diagrams. In [7] the authors describe an algorithm to generate concrete Euler diagrams from abstract diagrams. The layout of these automatically generated concrete diagrams can be improved using metrics to measure the layout quality and hill climbing techniques to improve the quality [10]. The work is extended in [34] to include embedding spiders in concrete Euler diagrams. The layout techniques have been extended to allow a given abstract diagram to be drawn in such a way that it appears similar to another concrete diagram [36]. This is important in automated theorem proving, since after applying a reasoning rule (which takes place at the abstract level) we wish the resulting automatically generated concrete diagram to appear similar to the premise.

9 Constraint diagrams

Introduced by Kent [29], constraint diagrams are designed for use by software engineers to specify formal constraints in object oriented systems. The constraint diagram in figure 23 specifies that no members are videos and for each member, all the past rentals, of which there is at least one, are videos. Formal mathematical methods have not been readily taken up by software en-

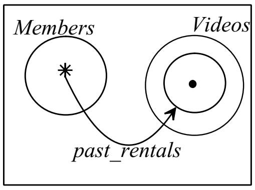

Fig. 23. Specifying software systems using constraint diagrams.

gineers because they tend to dislike using mathematical notations [29]. There is, however, a prevalent use of diagrams to model software systems.

The language of constraint diagrams extends that of spider diagrams by adding further syntactic elements, including *arrows*, *universal spiders* (represented by asterisks) and *derived contours* (contours that are not labelled). Arrows, together with their source, target and label, represent properties of binary relations. Universal spiders represent universal quantification. Derived contours must be the target of an arrow and represent the image of a relation when the domain is restricted to the source. Arrow sources can be spiders, contours and derived contours. Targets can be existential spiders, constant spiders, contours and derived contours. Constraint diagrams increase expressiveness over spider diagrams and are able to express the complex constraints that are required by software engineers.

Example 9.1 The constraint diagram in figure 24 expresses

$$\exists x \in A \bullet \{x\}.f = B \wedge A \cap B = \emptyset$$

where $\{x\}.f$ is the relational image of x under the relation f, that is

$$\{x\}.f = \{b : (a, b) \in (\{x\} \times U) \cap f\}.$$

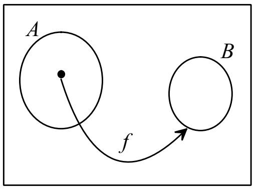

Fig. 24. A constraint diagram.

Example 9.2 The constraint diagram in figure 25 contains a universal spider and a derived contour. The diagram expresses

$$\forall x \in A \bullet \{x\}.f \subseteq U - A$$

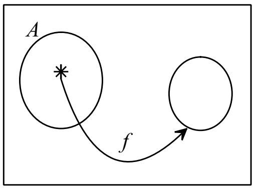

Fig. 25. A constraint diagram containing a universal spider.

The constraint diagrams in the two examples above are unambiguous and easy to interpret. One difficultly that arises when attempting to formalize constraint diagrams relates to the ordering of quantifiers. The non-linearity of diagrammatic notations gives rise to these ordering issues: there is often no clear starting point to interpret a diagram and even once reading has 'started' there are often further choices in reading order to be made, giving rise to ambiguities. Many of these difficulties are raised in [12] and further discussed in [14].

Example 9.3 In figure 26, the constraint diagram has two possible interpretations:

$$\forall x \in A\, \exists y \in U - A \bullet \{x\}.f = \{y\}$$

and

$$\exists y \in U - A\, \forall x \in A \bullet \{x\}.f = \{y\}.$$

These are not semantically equivalent. For example, the first interpretation, where the universal spider is read before the existential spider, has an empty model[4] whereas the second interpretation does not.

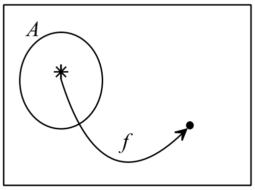

Fig. 26. Illustrating ambiguities: the ordering of quantifiers.

A *dependency analysis* is required to interpret a unitary diagram: the interpretation of certain syntactic components sometimes depends on first interpreting other syntactic components. In [4] a reading algorithm is given that involves a detailed and complex dependence analysis of the syntactic components of the diagram. This dependence analysis permits the specification of which syntactic components of the diagram require ordering when interpreting the diagram. A partially directed *dependence graph* is produced that indicates which syntactic elements are semantically dependent on each other. From the dependence graph for a diagram, *reading trees* can be produced and each tree gives rise to a semantic interpretation of the diagram. An algorithm for constructing all reading trees from a dependence graph and a diagram can be found in [5]. Thus, we can think of a constraint diagram as being a collection of zones (some shaded), spiders and arrows together with a reading tree.

Example 9.4 The diagram d in figure 27 has dependence graph $G(d)$. The spider s must be 'read' before the arrow labelled g and to indicate this there is a directed edge from s to g in $G(d)$. Similarly, t must be read before the arrow labelled f, giving a directed edge from t to f. The interpretations of t and the arrow labelled g are related (the derived contour that g targets defines the habitat of t), but we can choose whether to read t then g or g then t. In $G(d)$ this choice is indicated by an undirected edge between t and g. The dependence graph is used to construct reading trees, $R(d)$ for d. Two such reading trees are $R_1(d)$ and $R_2(d)$. If there is a directed edge from a to b in $G(d)$ then there must be a path from a to b in $R(d)$. If a and b are joined by an edge in $G(d)$ then there must be a path from a to b or from b to a. The reading tree also has a root node labelled PTC for the *plane tiling condition*. The plane tiling condition asserts that the union of the sets represented by the

[4] In most treatments of first order logic, the empty model is not permitted but in the application domain of constraint diagrams (i.e. software engineering) it is important to allow the empty model.

zones in d is the universal set. Reading tree $R(d_1)$ gives rise to the following semantic interpretation of d:

$$PTC \wedge \exists t \in U - A\ t.f = A \wedge \forall s \in A\ s.g \subseteq U - A \wedge t \in s.g.$$

Reading tree $R_2(d)$ gives rise to the semantic interpretation

$$PTC \wedge \forall s \in A\ s.g \subseteq U - A \wedge \exists t \in U - A\ t \in s.g \wedge t.f = A.$$

These two interpretations are not semantically equivalent.

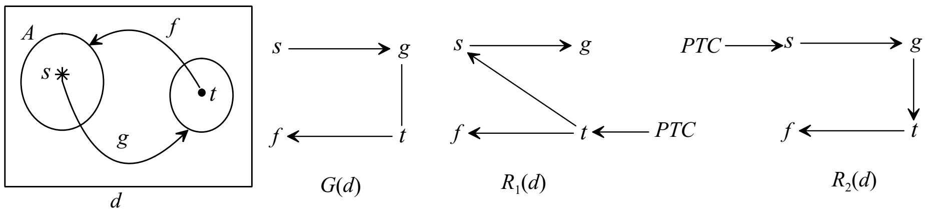

Fig. 27. A constraint diagram with its dependence graph and two reading trees.

In [6] the authors propose a default reading for constraint diagrams. In essence, this chooses a reading tree from the set of reading trees in such a way that the semantics are the 'natural' ones associated with the diagram.

Some sound (but not complete) reasoning rules have been developed for the full constraint diagram language [3]. The constraint diagram system CD1 introduced in [40,41] is both sound and complete. CD1 forms a decidable fragment of the full constraint diagram language and is restricted both in terms of syntax and semantics. Syntactic restrictions are placed on the sources and targets of arrows. Sources can only be spiders and targets can be existential spiders, contours and derived contours. Semantically, the exists is deemed to take precedence over for all. Furthermore, derived contours are restricted to representing the empty set. All the reasoning rules for SD3 generalize to CD1 and CD1 includes many further rules relating to arrows.

Example 9.5 The CD1 diagram d_1 in figure 28 asserts that

$$\exists x \in Stores\, \forall y \in Members \bullet \{y\}.Member\_of = \{x\} \wedge |Stores| = 1.$$

From this we can deduce

$$\exists x \in Stores\, \forall x \in Memebers \bullet \{y\}.Memeber\_of = Stores \wedge \\ \{y\}.Member\_of = \{x\} \wedge |Stores| = 1,$$

asserted by d_2.

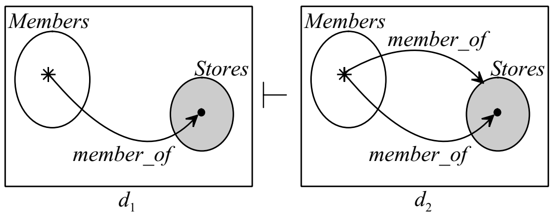

Fig. 28. Reasoning with CD1 constraint diagrams.

Like Venn-Peirce, Venn-II and Euler/Venn, CD1 diagrams can present contradictory information in a single (unitary) diagram. However, the way in which contradictions in single diagrams occur is rather different in CD1: through the use of arrows.

Example 9.6 The diagram in figure 29 asserts that there is an element, a, in A that is related to exactly one element, x, that is not in A under f and exactly one element, y, distinct from x, under f, which is a contradiction.

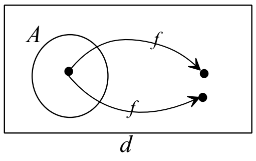

Fig. 29. An unsatisfiable CD1 diagram.

10 Conclusions and Future Work

In this paper we have outlined many reasoning systems that have emerged in recent years that extend Euler diagrams. Most of these systems are sound and complete. The relationship between the expressiveness of some of these systems is known, for example constraint diagrams are more expressive spider diagrams which, in turn, are more expressive than Venn-II diagrams. The most expressive language that we have discussed is the full constraint diagram notation, which is unlikely to be decidable (all of the other systems reviewed in this paper are decidable). The constraint diagram language is sufficiently expressive for practical application.

A goal is to develop sound and complete rules for the constraint diagram language and establish its exact expressive power. Such a task is challenging. One approach that can be taken to establish the expressive power of constraint diagrams, is to identify a fragment of first order predicate logic (FOPL) that is equivalent in expressive power to constraint diagrams. However, due to the vastly differing syntax of these two languages, identifying such a fragment of FOPL is a non-trivial task. Indeed, there may be no 'nice' fragment. A further goal is to develop software tools to support modelling and reasoning with constraint diagrams.

References

[1] L. Choudhury and M. K. Chakraborty. On extending Venn diagrams by augmenting names of individuals. In *Proceedings of Diagrams 2004, Cambridge, UK*, LNAI, pages 142–146. Springer-Verlag, March 2004.

[2] L. Euler. Lettres a une princesse d'allemagne. Sur divers sujets de physique et de philosophie, 1761. Letters No. 102-108 vol 2 Basel, Birkhauser.

[3] A. Fish and J. Flower. Investigating reasoning with constraint diagrams. In *Visual Language and Formal Methods*, ENTCS, pages 53–67, Rome, Italy, 2004. Elsevier.

[4] A. Fish, J. Flower, and J. Howse. A reading algorithm for constraint diagrams. In *IEEE Symposium on Human Centric Computing Languages and Environments, Auckland, New Zealand*, pages 116–168. IEEE, September 2003.

[5] A. Fish and J. Howse. Computing reading trees for constraint diagrams. In *AGTIVE '03, Applications of Graph Transformations with Industrial Relevance, Charlottesville, Virginia, September*, pages 260–274. Springer-Verlag, 2003.

[6] A. Fish and J. Howse. Towards a default reading for constraint diagrams. In *Proceedings of Diagrams 2004, Cambridge, UK*, LNAI, pages 51–65. Springer-Verlag, March 2004.

[7] J. Flower and J. Howse. Generating Euler diagrams. In *Proceedings of Diagrams 2002, Callaway Gardens, Georgia, USA*, pages 61–75. Springer-Verlag, April 2002.

[8] J. Flower, J. Masthoff, and G. Stapleton. Generating proofs with spider diagrams using heuristics. In *Proceedings of Distributed Multimedia Systems, International Workshop on Visual Languages and Computing*, pages 279–285. Knowledge Systems Institute, 2004.

[9] J. Flower, J. Masthoff, and G. Stapleton. Generating readable proofs: A heuristic approach to theorem proving with spider diagrams. In *Proceedings of 3rd International Conference, Diagrams 2004, Cambridge, UK*, pages 166–181. Springer-Verlag, 2004.

[10] J. Flower, P. Rodgers, and P. Mutton. Layout metrics for Euler diagrams. In *7th International Conference on Information Visualisation*, pages 272–280. IEEE Computer Society Press, 2003.

[11] J. Flower and G. Stapleton. Automated theorem proving with spider diagrams. In *Proceedings of Computing: The Australasian Theory Symposium (CATS'04), Dunedin, New Zealand*, volume 91 of *ENTCS*, pages 116–132. Science Direct, January 2004.

[12] J. Gil, J. Howse, and S. Kent. Constraint diagrams: A step beyond UML. In *Proceedings of TOOLS USA 1999, Santa Barbara, California, USA*, pages 453–463. IEEE Computer Science Press, August 1999.

[13] J. Gil, J. Howse, and S. Kent. Formalising spider diagrams. In *Proceedings of IEEE Symposium on Visual Languages (VL99), Tokyo*, pages 130–137. IEEE Computer Society Press, September 1999.

[14] J. Gil, J. Howse, and S. Kent. Towards a formalization of constraint diagrams. In *Proc IEEE Symposia on Human-Centric Computing (HCC '01), Stresa, Italy*, pages 72–79. IEEE Computer Society Press, September 2001.

[15] J. Gil, J. Howse, S. Kent, and J. Taylor. Projections in Venn-Euler diagrams. In *Proc. IEEE Symposium on Visual Languages*, pages 119–126. IEEE Computer Society Press, September 2000.

[16] J. Gil, J. Howse, and E. Tulchinsky. Positive semantics of projections. *Journal of Visual Languages and Computing*, 13(2):197–227, April 2001.

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

[18] E. Hammer and N. Danner. Towards a model theory of Venn diagrams. In G. Allwein and J. Barwise, editors, *Logical Reasoning with Diagrams*, pages 109–127. Oxford University Press, 1996.

[19] E. Hammer and S. J. Shin. Euler's visual logic. *History and Philosophy of Logic*, pages 1–29, 1998.

[20] J. Howse, F. Molina, S.-J. Shin, and J. Taylor. Type-syntax and token-syntax in diagrammatic systems. In *Proceedings FOIS-2001: 2nd International Conference on Formal Ontology in Information Systems, Maine, USA*, pages 174–185. ACM Press, 2001.

[21] J. Howse, F. Molina, S.-J. Shin, and J. Taylor. On diagram tokens and types. In *Proceedings of Diagrams 2002, Callaway Gardens, Georgia, USA*, pages 146–160. Springer-Verlag, April 2002.

[22] J. Howse, F. Molina, and J. Taylor. On the completeness and expressiveness of spider diagram systems. In *Proceedings of Diagrams 2000, Edinburgh, UK*, pages 26–41. Springer-Verlag, September 2000.

[23] J. Howse, F. Molina, and J. Taylor. SD2: A sound and complete diagrammatic reasoning system. In *Proceedings VL 2000: IEEE Symposium on Visual Languages, Seattle, USA*, pages 127–136. IEEE Computer Society Press, 2000.

[24] J. Howse, F. Molina, and J. Taylor. A sound and complete diagrammatic reasoning system. In *Proceedings. ASC 2000: 3rd IASTED International Conference on Artificial Intelligence and Soft Computing, Banff*, pages 402–408. IASTED/ACTA Press, 2000.

[25] J. Howse, F. Molina, J. Taylor, and S. Kent. Reasoning with spider diagrams. In *Proceedings of IEEE Symposium on Visual Languages (VL99), Tokyo*, pages 138–147. IEEE Computer Society Press, September.

[26] J. Howse, F. Molina, J. Taylor, S. Kent, and J. Gil. Spider diagrams: A diagrammatic reasoning system. *Journal of Visual Languages and Computing*, 12(3):299–324, June 2001.

[27] J. Howse, G. Stapleton, and J. Taylor. Reasoning with spider diagrams. In *Available from www.cmis.brighton.ac.uk/research/vmg.*, 2004.

[28] C. John. Reasoning with projected contours. In *Proceedings of 3rd International Conference, Diagrams 2004, Cambridge, UK*, pages 147–150. Springer-Verlag, 2004.

[29] S. Kent. Constraint diagrams: Visualizing invariants in object oriented modelling. In *Proceedings of OOPSLA97*, pages 327–341. ACM Press, October 1997.

[30] O. Lemon. Comparing the efficacy of visual languages. In D. Barker-Plummer, D. I. Beaver, J. van Benthem, and P. Scotto di Luzio, editors, *Words, Proofs and Diagrams*, pages 47–69. CSLI Publications, 2002.

[31] O. Lemon and I. Pratt. Spatial logic and the complexity of diagrammatic reasoning. *Machine GRAPHICS and VISION*, 6(1):89–108, 1997.

[32] F. Molina. *Reasoning with extended Venn-Peirce diagrammatic systems.* PhD thesis, University of Brighton, 2001.

[33] T. More. On the construction of Venn diagrams. *Journal of Symbolic Logic*, 23:303–304, 1959.

[34] P. Mutton, P. Rodgers, and J. Flower. Drawing graphs in Euler diagrams. In *Proceedings of 3rd International Conference, Diagrams 2004, Cambridge, UK*, pages 66–81. Springer-Verlag, 2004.

[35] C. Peirce. *Collected Papers*, volume 4. Harvard University Press, 1933.

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

[37] F. Ruskey. A survey of Venn diagrams. *Electronic Journal of Combinatorics*, 1997. www.combinatorics.org/Surveys/ds5/VennEJC.html.

[38] P. Scotto di Luzio. Patching up a logic of Venn diagrams. In *Proceedings 6th CSLI WS on Logic, Language and Computation*. CSLI Publications, 2000.

[39] S.-J. Shin. *The Logical Status of Diagrams*. Cambridge University Press, 1994.

[40] G. Stapleton. *Reasoning with Constraint Diagrams*. PhD thesis, School of Computing, Mathematical and Information Sciences, August 2004.

[41] G. Stapleton, J. Howse, and J. Taylor. A constraint diagram reasoning system. In *Proceedings of International Conference on Visual Languages and Computing*, pages 263–270. Knowledge Systems Insitute, 2003.

[42] G. Stapleton, J. Howse, J. Taylor, and S. Thompson. The expressiveness of spider diagrams augmented with constants. In *Visual Languages and Human Centric Computing, Rome, Italy*, pages 91–98. IEEE Computer Society Press, September 2004.

[43] G. Stapleton, J. Howse, J. Taylor, and S. Thompson. What can spider diagrams say? In *Proceedings of 3rd International Conference, Diagrams 2004, Cambridge, UK*, pages 112–127. Springer-Verlag, 2004.

[44] G. Stapleton, S. Thompson, J. Howse, and J. Taylor. The expressiveness of spider diagrams. *Journal of Logic and Computation*, 14(6):857–880, December 2004.

[45] N. Swoboda. Implementing Euler/Venn reasoning systems. In M. Anderson, B. Meyer, and P. Olivier, editors, *Diagrammatic Representation and Reasoning*, pages 371–386. Springer-Verlag, 2001.

[46] N. Swoboda and G. Allwein. A case study of the design and implementation of heterogeneous reasoning systems. In L. Magnani, N. J. Neressian, and C. Pizzi, editors, *Logical and Computational Aspects of Model-Based Reasoning*, pages 1–18. Kluwer Academic Publishers, 2002.

[47] N. Swoboda and G. Allwein. Modeling heterogeneous systems. In *Proceedings of Diagrams 2002, Callaway Gardens, Georgia, USA*, pages 131–145. Springer-Verlag, 2002.

[48] N. Swoboda and G. Allwein. Using DAG transformations to verify Euler/Venn homogeneous and Euler/Venn FOL heterogeneous rules of interence. In *Proceedings of GT-VMT*, ENTCS. Elsevier Science, 2002.

[49] N. Swoboda and J. Barwise. The information content of Euler/Venn diagrams. In *Proceedings LICS workshop on Logic and Diagrammatic Information*, 1998.

[50] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag*, 1880.

[51] J. Venn. *Symbolic Logic*. New York:Burt Franklin, 1971.

[52] A. Verroust and M.-L. Viaud. Ensuring the drawability of Euler diagrams for up to eight sets. In *Proceedings of 3rd International Conference, Diagrams 2004, Cambridge, UK*, LNAI, pages 128–141. Springer-Verlag, March 2004.