Type-syntax and Token-syntax in Diagrammatic Systems

John Howse
School of Computing & Mathematics
University of Brighton, Brighton, UK
`John.Howse@brighton.ac.uk`

Fernando Molina
School of Computing & Mathematics
University of Brighton, Brighton, UK
`F.Molina@brighton.ac.uk`

Sun-Joo Shin
Department of Philosophy
University of Notre Dame,
Notre Dame, Indiana, USA
`Sun-Joo.Shin.3@nd.edu`

John Taylor
School of Computing & Mathematics
University of Brighton, Brighton, UK
`John.Taylor@brighton.ac.uk`

Abstract

The uptake in the software industry of notations for designing systems visually has been accelerated with the standardization of the Unified Modeling Language (UML). The formalization of diagrammatic notations is important for the development of essential tool support and to allow reasoning to take place at the diagrammatic level. Focusing on an extended version of Venn and Euler diagrams (which was developed to complement UML in the specification of software systems), this paper presents two levels of syntax for this system: type-syntax and token-syntax. Token-syntax is about particular diagrams instantiated on some physical medium, and type-syntax provides a formal definition with which a concrete representation of a diagram must comply. While these two levels of syntax are closely related to each other, the domains of type-syntax and token-syntax are ontologically independent from each other, that is, one is abstract and the other concrete. We discuss the roles of type-syntax and token-syntax in diagrammatic systems and show that it is important to consider both levels of syntax in diagrammatic reasoning systems and in developing software tools to support such systems.

Keywords Visual formalisms, software specification, formal methods, concrete and abstract syntax, diagrammatic reasoning.

1 Introduction

Circles or closed curves, which we call contours, have been in use for the representation of classical syllogisms since at least the Middle Ages [16]. Euler introduced the notation we now call *Euler circles* (or *Euler diagrams*) [1] to illustrate relations between sets. This notation uses the topological properties of enclosure, exclusion and intersection to represent the set-theoretic notions of subset, disjointness, and intersection, respectively.

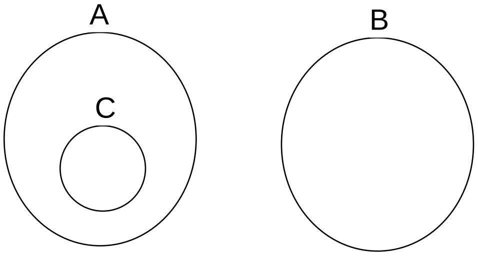

Fig. 1: An Euler diagram.

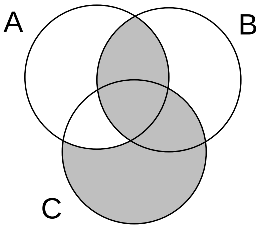

Fig. 2: A Venn diagram.

The logician John Venn used contours to represent logical propositions [23]. In Venn diagrams all contours must intersect. Moreover, for each non-empty subset of the contours, there must be a connected region of the diagram, such that the regions enclosed by the contours in this subset intersect at exactly that region. Shading is then used to show that a particular region of the resulting map is empty. The Euler diagram in Fig. 1 denotes that A and B are disjoint and $C \subseteq A$. A Venn diagram showing the same information is given in Fig. 2.

The logician Charles Peirce augmented Venn diagrams by adding *X-sequences* as a means for denoting elements [20]. An X-sequence connecting a number of "minimal regions" of a Venn diagram, indicates that their union

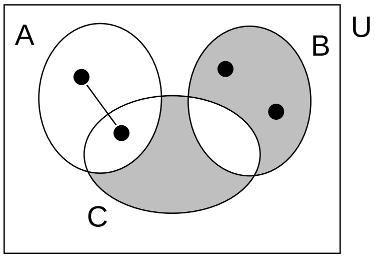

Fig. 3: A Spider diagram.

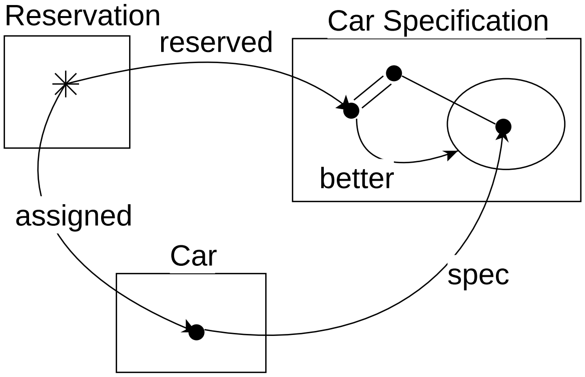

Fig. 4: A constraint diagram.

is not empty. Full semantics and sound and complete inference rules have been developed for Venn-Peirce diagrams [22] and Euler circles [9].

Spider diagrams [3] are a natural extension of Venn-Peirce and Euler diagrams; they are based on Euler diagrams, so the topological properties of the diagrams are important, but they also contain *spiders*, a generalization of Peirce's X-sequences, and shading. The spider diagram in Fig. 3 denotes that A and B are disjoint, there are no elements in C that are not in A or B, there is an element in A and there are exactly two elements in B that are not in A or C.

Spider diagrams emerged from work on *constraint diagrams* [15], introduced as a visual technique intended to be used in conjunction with the Unified Modeling Language (UML) [19] for object-oriented modelling. The constraint diagram in Fig. 4 expresses, among other constraints, an invariant on a model of a car-hire business: *The specification of the car assigned to a reservation must be the same or better than the specification reserved.*

$$\forall r \in Reservations, r.assigned.spec = r.reserved \\ \vee\ r.assigned.spec \in r.reserved.better.$$

Currently in UML, such constraints can only be expressed using the Object Constraint Language (OCL) [24], essentially a stylized, textual version of first-order predicate logic.

Focusing on a subset of spider diagram notation, this paper presents two levels of syntax for this system: *type-syntax* and *token-syntax*. Token-syntax is about particular diagrams instantiated on some physical medium, and type-syntax provides a formal definition with which a concrete representation of a diagram must comply. While these two levels of syntax are closely related to each other, the domains of type-syntax and token-syntax are ontologically independent from each other, that is, one is abstract and the other concrete. This paper is *not* concerned with the semantics of the notation, although occasionally semantic aspects will arise. A concise informal description of spider diagrams is given in §2. In §3 we explore ontological issues of diagrammatic languages to argue for the necessity of two levels of syntax for a diagrammatic system. In §4 we give formal definitions of the type-syntax and token-syntax of spider diagrams and the mappings between them and consider equivalent diagrams. In §5 we discuss the utility of a two-tiered syntax, including diagrammatic reasoning and the use of software tools. §6 summarizes the paper and discusses further work.

2 Spider diagrams

In this section we give a concise informal description of spider diagrams. A *contour* is a simple closed plane curve. A *boundary rectangle* properly contains all other contours, although we do not always represent it in concrete diagrams; its existence is implicit. A *district* (or *basic region*) is the bounded area of the plane enclosed by a contour or by the boundary rectangle. A *region* is defined, recursively, as follows: any district is a region; if r_1 and r_2 are regions, then the union, intersection, or difference, of r_1 and r_2 are regions provided these are non-empty. A *zone* (or *minimal region*) is a region having no other region contained within it. Contours and regions denote sets.

A *spider* is a tree with nodes (called *feet*) placed in different zones; the connecting edges (called *legs*) are straight lines; a spider with legs is said to be *articulated*. A spider *touches* a zone if one of its feet appears in that zone. A spider may touch a zone at most once. A spider is said to *inhabit* the region which is the union of the zones it touches. For any spider s, the *habitat* of s, denoted $\eta(s)$, is the region inhabited by s. A spider corresponds to existential quantification. Two distinct spiders denote distinct elements.

Every region is a union of zones. A region is *shaded* if each of its component zones is shaded. The semantics of a *shaded zone* is that the set it denotes may not contain elements other than those indicated by the spiders which touch that zone. Hence, a shaded region denotes the empty set if it is not touched by any spider. Spiders can be used to place a lower bound on the number of elements in a set; shading a zone which includes spiders has the effect of placing an upper bound on the cardinality of the set denoted by that zone. Each contour must be labelled and no two contours can have the same label.

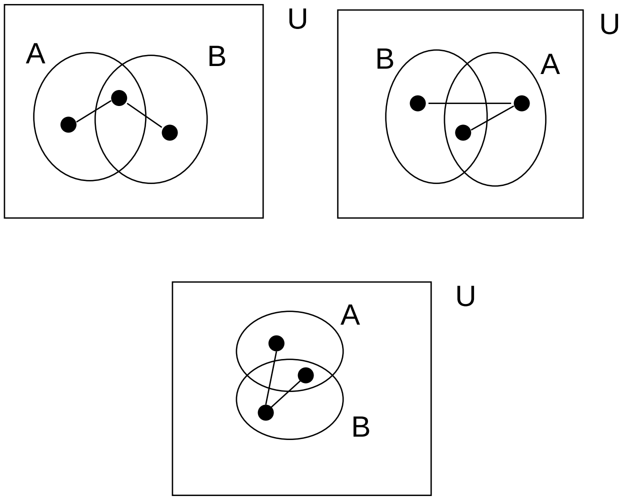

Fig. 5: Equivalent diagrams.

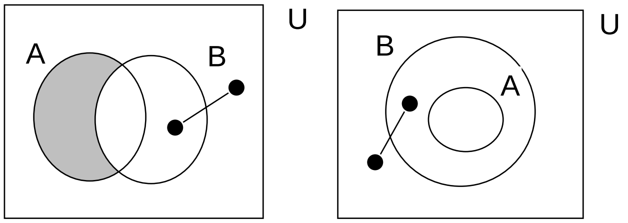

Fig. 6: Semantically-equivalent diagrams.

The spider diagram in Fig. 3 has a boundary rectangle and three contours and hence four labels; six zones, two of which are shaded, and hence 63 regions and three shaded regions; and three spiders, one of which is articulated.

In this paper, we consider only a subset of the full notation. Spider diagrams also contain notation to represent given and optional elements and for saying that two elements can or must be equal. For more details see [3]. Spider diagrams include the idea of *projections*, which remove clutter and focus attention in the diagram appropriately; for details see [4, 5, 6]. Conjunctive and disjunctive systems of spider diagrams have been developed and sound and complete diagrammatic inference rules have been developed for several systems of spider diagrams [10, 11, 12, 13, 14, 17].

3 Two-tiered ontology

In this section, we explore ontological issues of diagrammatic languages to argue for the necessity of two levels of syntax for a diagrammatic system. Consider the three diagrams in Fig. 5. These are three different diagrams whose appearances are clearly distinguishable from one another. In the first diagram the contour labeled as “A” is to the left of the contour labeled as “B,” in the second the location of these two contours is switched, and in the third one the “B” contour is placed below the contour “A.” Also, all of the three diagrams exhibit different positions of their spiders’ legs. Hence, if the spatial arrangements of contours and spiders’ legs were representing facts, these diagrams would represent different facts. However, in the the spider diagram system, neither the location of contours nor the configuration of spiders has a representing import. Therefore, we would like to say that these diagrams are the “same” in *some* sense.

On the other hand, we do not want say that the two spider diagrams in Fig. 6 are the same even though they represent the same fact. The semantics of the spider diagram system tells us that these two diagrams represent the same fact. But, there is a clear syntactic distinction between two diagrams: among other differences, there is shading in the one diagram, but not in the other. That is, the difference and sameness in the case of Fig. 6 can be seen to reflect a distinction between syntactic difference and semantic equivalence.

An interesting task is how to capture our intuition about the diagrams in Fig. 5: They are different from one another in one sense, but the same in another sense. One way to approach this case is not to make a distinction between Fig. 5 and Fig. 6 and to conclude that the diagrams in Fig. 5 are syntactically different from one another but semantically the same. A problem with this method is that the syntax would became so fine-grained that almost any pair of diagrams cannot be said to be syntactically the same. That is, any burden for differentiation among diagrams is postponed to a semantic level. At an opposite extreme direction from this approach, another way to explain the relation among diagrams in Fig. 5 is to say that that these diagrams are all syntactically equivalent to one another. A problem with this explanation is that the syntax would become too coarse to accommodate differences among obviously present visual properties.

We present a middle ground solution between the two above alternatives, by bringing in a time-honored distinction between type and token [8, 20]: The diagrams in Fig. 5 are *different tokens*, but of the *same type*. Depending on whether we talk about a diagram as a token or as a type, we may attribute different properties to a diagram or different relations among diagrams. Therefore, it is crucial to disambiguate which ontological status of a diagram is in question, either a concrete token-diagram or an abstract type-diagram. We suggest that the syntax of a diagrammatic system consist of two different levels, that is, a token-level and a type-level. At the token-level the syntax is fine-grained enough to respect our intuition that the diagrams in Fig. 5 are *different*, while the other level of syntax, i.e. type-syntax, lets us say that these diagrams are the *same*. Furthermore, we claim that a fine-grained two-tiered syntax is not only desirable but also almost necessary for a diagrammatic system because failure to recognize two different ontological statuses of diagrammatic language, i.e.

token and type, could cause a significant ambiguity in our investigation of the system.

Quite interestingly, a similar confusion between type and token does not cause much trouble for symbolic systems. The conventions of linear symbolic systems are so uniform that we may safely ignore token-level syntax. That is, when two tokens belong to the same type, visual differences among these two tokens are trivial. Hence, an ambiguity between sentence-token (or symbol-token) and sentence-type (or symbol-type), if any, is negligible. For example, in the following example, sentence (1) and sentence (2) are two different sentence-tokens but are of the same type, while sentence (2) and sentence (3) are not only different tokens but belong to different types:

(1) ∀xP(x) ∨ ∃yP(y)
(2) $\forall x P(x) \vee \exists y P(y)$
(3) $\exists y P(y) \vee \forall x P(x)$

In this case, the related conventions are clear: Neither the size nor the font of a symbol is a representing fact, while linear order is. Therefore, the first two sentences belong to the same type, while (3) does not. These conventions are so ingrained that we almost do not see a visual difference between (1) and (2).

On the other hand, we need more detailed knowledge of the spider diagram system to find out whether diagrams in Fig. 5 are of the same type. At the same time, visual differences among the three diagrams are clear enough to think intuitively that they are different in some sense. In the next section, we develop two-tiered syntax for the spider diagram system to take care of a type-token distinction which is unique to diagrammatic systems.

4 Type-syntax and token-syntax

The **type-syntax** (or *abstract* syntax) of a spider diagram d is a formal definition that is independent of any concrete visual representation. A concrete **instantiation** of d is a diagram presented on some physical medium (e.g., a sheet of paper, a computer monitor, etc) that *complies* with the abstract definition; we call this the **token-syntax**. In this section we give the type-syntax of spider diagrams and a formal definition of the token-syntax. We also give conditions for a concrete diagram to be an instantiation of an abstract diagram and for an abstract diagram to be an abstraction of a concrete diagram.

4.1 Type-syntax

An *abstract spider diagram* is a tuple

$$d = \langle \mathcal{C}, \beta, \mathcal{Z}, \mathcal{Z}^*, \mathcal{L}, \ell, \mathcal{S}, \eta \rangle$$

whose components are defined as follows:

(i) $\mathcal{C}$ is a finite set whose members are called *contours*. The element β, which is not a member of $\mathcal{C}$, is called the *boundary rectangle*.

(ii) The set $\mathcal{Z} \subseteq 2^{\mathcal{C}}$ is the set of *zones*, while $\mathcal{Z}^* \subseteq \mathcal{Z}$ is the set of *shaded zones*. A zone $z \in \mathcal{Z}$ is *incident* on a contour $c \in \mathcal{C}$ if $c \in z$. Let $\mathcal{R} = 2^{\mathcal{Z}} - \emptyset$ be the set of *regions*, and $\mathcal{R}^* = 2^{\mathcal{Z}^*}$ be the set of shaded regions.

(iii) $\mathcal{L}$ is the set of contour *labels*. The bijection $\ell : \mathcal{C} \cup \{\beta\} \rightarrow \mathcal{L}$ returns the label of a contour or the boundary rectangle.

(iv) $\mathcal{S}$ is a set of *spiders*.

(v) The function $\eta : \mathcal{S} \rightarrow \mathcal{R}$ returns the *habitat* of a spider.

A zone is defined by the contours that contain it and is thus represented as a set of contours. Not all possible zones need appear on a spider diagram as the underlying diagram is an Euler diagram; if they all do appear, then the diagram is in Venn form. A region is just a non-empty set of zones.

4.2 Token-syntax

A *concrete spider diagram* $\hat{d}$ is a tuple

$$\hat{d} = \langle \hat{\mathcal{C}}, \hat{\beta}, \hat{\mathcal{Z}}, \hat{\mathcal{Z}}^*, \hat{\mathcal{L}}, \hat{\ell}, \hat{\mathcal{S}}, \hat{\eta} \rangle$$

whose components are defined as follows:

(i) $\hat{\mathcal{C}}$ is a finite set of simple closed (Jordan) curves in the plane, $\mathbf{R}^2$, called *contours*. The *boundary rectangle*, $\hat{\beta}$, is also a simple closed curve, usually in the form of a rectangle, but not a member of $\hat{\mathcal{C}}$. For any contour $\hat{c}$ (including $\hat{\beta}$) we denote by $\iota(\hat{c})$ and $\varepsilon(\hat{c})$ the interior (bounded) and the exterior (unbounded) components of $\mathbf{R}^2 - \hat{c}$ respectively; such components exist by the *Jordan Curve Theorem*. Each contour lies within, and does not touch, the boundary rectangle: $\hat{c} \subset \iota(\hat{\beta})$. The set $\hat{\mathcal{C}}$ forms an *Euler diagram* which has the following properties:

1. Contours intersect transversely.
2. Each contour intersects with every other contour an even number of times. This can, of course, be zero times.
3. No two contours have a point in common without crossing at that point.
4. Each component of $\mathbf{R}^2 - \bigcup\limits_{\hat{c} \in \hat{\mathcal{C}}} \hat{c}$ is the intersection of $\iota(\hat{c})$ for all contours $\hat{c}$ in some subset X of $\hat{\mathcal{C}}$ and $\varepsilon(\hat{c})$ for all contours $\hat{c}$ in the complement of X:

$$\bigcap_{\hat{c} \in X} \iota(\hat{c}) \cap \bigcap_{\hat{c} \in \hat{\mathcal{C}} - X} \varepsilon(\hat{c}).$$

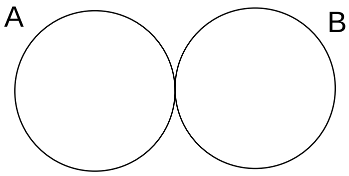

Fig. 7: Touching contours.

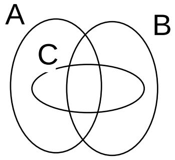

Fig. 8: A disconnected zone.

(ii) A *zone* is the intersection of a component of $\mathbf{R}^2 - \bigcup_{\hat{c}\in\hat{\mathcal{C}}} \hat{c}$ with $\iota(\hat{\beta})$. $\hat{\mathcal{Z}}$ is the set of zones. $\hat{\mathcal{Z}}^*$ is the set of shaded zones. Let $\hat{\mathcal{R}} = 2^{\hat{\mathcal{Z}}} - \emptyset$ be the set of *regions*, and $\hat{\mathcal{R}}^* = 2^{\hat{\mathcal{Z}}^*}$ be the set of shaded regions.

(iii) $\hat{\mathcal{L}}$ is the set of contour labels. The bijection $\hat{\ell} : \hat{\mathcal{C}} \cup \{\hat{\beta}\} \rightarrow \hat{\mathcal{L}}$ returns the label of a contour or the boundary rectangle.

(iv) $\hat{\mathcal{S}}$ is a finite set of plane trees, called *spiders*, whose nodes, called feet and represented by filled-in circles, lie within $\hat{\mathcal{Z}}$ and satisfy the following properties:

1. each spider has at most one foot in each zone;
2. the edges of each spider are straight line segments;
3. no two spiders have a foot in common.

(v) $\hat{\eta} : \hat{\mathcal{S}} \rightarrow \hat{\mathcal{R}},\ \hat{\eta}(\hat{s}) = \{\hat{z} \in \hat{\mathcal{Z}} | \hat{s}$ has a foot in $\hat{z}\}$.

Hammer [9] defines an Euler diagram as 'any finite number of closed curves drawn on the page in any arrangement'. This is, of course, a very liberal definition. We have chosen to ban some possible Euler diagrams for the sake of intuitive clarity. For example the diagram in Fig. 7 in which two contours touch but do not cross is not well-formed as are diagrams with disconnected zones, an example of which is given in Fig. 8, which is attempting to show that $C \subseteq A \cup B$, the zone $\{A, B\}$, i.e., that part of the diagram within A and B, but outside C, is disconnected. Allowing disconnected zones could cause intuitive problems in interpreting the diagram; it would also allow all sorts of strange diagrams.

An alternative way to represent $C \subseteq A \cup B$ is given in Fig. 9. In this diagram three contours intersect at a point, but all zones are connected.

4.3 Mappings between diagram types and tokens

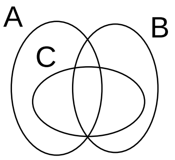

Fig. 9: A triple point.

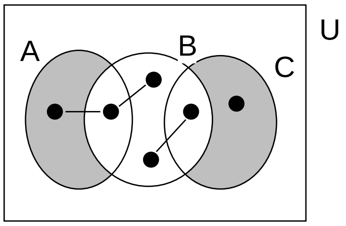

Fig. 10: A concrete spider diagram.

Let $\hat{d} = \langle \hat{\mathcal{C}}, \hat{\beta}, \hat{\mathcal{Z}}, \hat{\mathcal{Z}}^*, \hat{\mathcal{L}}, \hat{\ell}, \hat{\mathcal{S}}, \hat{\eta} \rangle$ be a concrete diagram and let $d = \langle \mathcal{C}, \beta, \mathcal{Z}, \mathcal{Z}^*, \mathcal{L}, \ell, \mathcal{S}, \eta \rangle$ be an abstract diagram. Then d is an **abstraction** of $\hat{d}$ if there is a mapping $\mu : \hat{d} \rightarrow d$ such that component mappings $\mu : \hat{\mathcal{C}} \rightarrow \mathcal{C}$, $\mu : \hat{\mathcal{Z}} \rightarrow \mathcal{Z}$, $\mu : \hat{\mathcal{L}} \rightarrow \mathcal{L}$, $\mu : \hat{\mathcal{S}} \rightarrow \mathcal{S}$ are each bijections and satisfy the following conditions:

1. $\forall \hat{c} \in \hat{\mathcal{C}}\, \forall c \in \mathcal{C} \bullet \mu(\hat{c}) = c \Leftrightarrow \mu(\hat{\ell}(\hat{c})) = \ell(c)$.
2. $\forall \hat{z} \in \hat{\mathcal{Z}} \bullet \mu(\hat{z}) = \{\mu(\hat{c}) \mid \hat{z} \subseteq \iota(\hat{c})\}$.
3. $\forall \hat{z} \in \hat{\mathcal{Z}} \bullet \mu(\hat{z}) \in \mathcal{Z}^* \Leftrightarrow \hat{z} \in \hat{\mathcal{Z}}^*$.
4. $\forall \hat{s} \in \hat{\mathcal{S}} \bullet \mu(\hat{\eta}(\hat{s})) = \eta(\mu(\hat{s}))$.

Such a mapping μ is said to be an *abstraction* mapping.

Similarly, $\hat{d}$ is a (concrete) **instantiation** of d if there is a mapping $\zeta : d \rightarrow \hat{d}$ such that component mappings $\zeta : \mathcal{C} \rightarrow \hat{\mathcal{C}}$, $\zeta : \mathcal{Z} \rightarrow \hat{\mathcal{Z}}$, $\zeta : \mathcal{L} \rightarrow \hat{\mathcal{L}}$, $\zeta : \mathcal{S} \rightarrow \hat{\mathcal{S}}$ are each bijections and satisfy the following conditions:

1. $\forall \hat{c} \in \hat{\mathcal{C}}\, \forall c \in \mathcal{C} \bullet \zeta(c) = \hat{c} \Leftrightarrow \zeta(\ell(c)) = \hat{\ell}(\hat{c})$.
2. $\forall z \in \mathcal{Z} \bullet$
$$\zeta(z) = \bigcap_{c \in z} \iota(\zeta(c)) \cap \bigcap_{c \in \mathcal{C} - z} \varepsilon(\zeta(c)) \cap \iota(\hat{\beta}).$$
3. $\forall z \in \mathcal{Z} \bullet \zeta(z) \in \hat{\mathcal{Z}}^* \Leftrightarrow z \in \mathcal{Z}^*$.
4. $\forall s \in \mathcal{S} \bullet \zeta(\eta(s)) = \hat{\eta}(\zeta(s))$.

Such a mapping ζ is said to be an *instantiation* mapping.

The spider diagram in Fig. 10 is an instantiation of the abstract diagram

$$d = \langle \mathcal{C}, \beta, \mathcal{Z}, \mathcal{Z}^*, \mathcal{L}, \ell, \mathcal{S}, \eta \rangle$$

where

$$\mathcal{C} = \{c_1, c_2, c_3\}$$

$$\mathcal{Z} = \{\emptyset, \{c_1\}, \{c_1, c_2\}, \{c_2\}, \{c_2, c_3\}, \{c_3\}\}$$

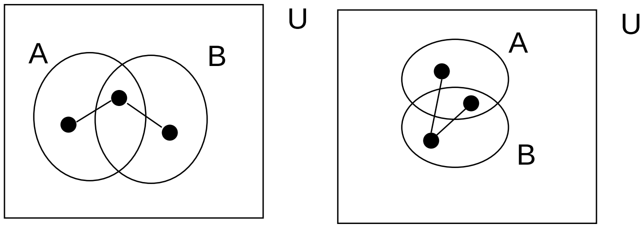

Fig. 11: Equivalent concrete diagrams.

$$\mathcal{Z}^* = \{\{c_1\}, \{c_3\}\}$$

$$\mathcal{L} = \{A, B, C\}$$

$$\ell(\beta) = U, \ell(c_1) = A, \ell(c_2) = B, \ell(c_3) = C$$

$$\mathcal{S} = \{s_1, s_2, s_3\}$$

$$\eta(s_1) = \{\{c_1\}, \{c_1, c_2\}, \{c_2\}\},$$

$$\eta(s_2) = \{\{c_2\}, \{c_2, c_3\}\}, \eta(s_3) = \{\{c_3\}\}.$$

Theorem 1 *Let $\hat{d}$ be a concrete diagram and d be an abstract diagram. Let $\mu : \hat{d} \to d$ be an abstraction mapping and $\zeta : d \to \hat{d}$ be an instantiation mapping. Then μ^{-1} is an instantiation mapping and ζ^{-1} is an abstraction mapping.*

The proof is omitted. We say that a concrete diagram *complies* with any of its abstractions and, equivalently, any instantiation of an abstract diagram complies with the abstract diagram.

4.4 Equivalent diagrams

For each abstract diagram there are many concrete instantiations. For example, the diagrams in Fig. 11 are both concrete representations of the abstract diagram

$$d_1 = \langle \mathcal{C}_1, \beta_1, \mathcal{Z}_1, \mathcal{Z}_1^*, \mathcal{L}_1, \ell_1, \mathcal{S}_1, \eta_1 \rangle$$

where

$$\mathcal{C}_1 = \{c_1, c_2\}$$

$$\mathcal{Z}_1 = \{\emptyset, \{c_1\}, \{c_1, c_2\}, \{c_2\}\}$$

$$\mathcal{Z}_1^* = \emptyset$$

$$\mathcal{L}_1 = \{A, B\}$$

$$\ell(\beta_1) = U, \ell_1(c_1) = A, \ell_1(c_2) = B$$

$$\mathcal{S}_1 = \{s_1\}$$

$$\eta_1(s_1) = \{\{c_1\}, \{c_1, c_2\}, \{c_2\}\}.$$

More surprisingly, a concrete diagram can have many abstractions. For example, each diagram in Fig. 11 is also a concrete instantiation of the abstract diagram

$$d_2 = \langle \mathcal{C}_2, \beta_2, \mathcal{Z}_2, \mathcal{Z}_2^*, \mathcal{L}_2, \ell_2, \mathcal{S}_2, \eta_2 \rangle$$

where

$$\mathcal{C}_2 = \{c_3, c_4\}$$

$$\mathcal{Z}_2 = \{\emptyset, \{c_3\}, \{c_3, c_4\}, \{c_4\}\}$$

$$\mathcal{Z}^* = \emptyset$$

$$\mathcal{L}_2 = \{A, B\}$$

$$\ell(\beta_2) = U, \ell_2(c_3) = A, \ell(c_4) = B$$

$$\mathcal{S}_2 = \{s_2\}$$

$$\eta_2(s_2) = \{\{c_3\}, \{c_3, c_4\}, \{c_4\}\}.$$

Clearly, the two abstractions d_1 and d_2 are 'isomorphic' in some sense. We now make this notion more precise.

Two abstract diagrams

$$d_1 = \langle \mathcal{C}_1, \beta_1, \mathcal{Z}_1, \mathcal{Z}_1^*, \mathcal{L}_1, \ell_1, \mathcal{S}_1, \eta_1 \rangle$$

and

$$d_2 = \langle \mathcal{C}_2, \beta_2, \mathcal{Z}_2, \mathcal{Z}_2^*, \mathcal{L}_2, \ell_2, \mathcal{S}_2, \eta_2 \rangle$$

are **isomorphic** if there is a mapping $\theta : d_1 \to d_2$ such that component mappings $\theta : \mathcal{C}_1 \to \mathcal{C}_2,\ \theta : \mathcal{Z}_1 \to \mathcal{Z}_2,\ \theta : \mathcal{L}_1 \to \mathcal{L}_2,\ \theta : \mathcal{S}_1 \to \mathcal{S}_2$ are each bijections and satisfy the following conditions:

1. $\forall c \in \mathcal{C}_1 \bullet \theta(\ell_1(c)) = \ell_2(\theta(c))$ and $\theta(\ell_1(\beta_1)) = \ell_2(\theta(\beta_2))$.
2. $\forall z \in \mathcal{Z}_1 \bullet z \in \mathcal{Z}_1^* \Leftrightarrow \theta(z) \in \mathcal{Z}_2^*$.
3. $\forall s \in \mathcal{S}_1 \bullet \theta(\eta_1(s)) = \eta_2(\theta(s))$.

In fact, this is precisely the algebraic notion of isomorphism when d_1 and d_2 are regarded as many-sorted algebras.

Two abstract diagrams d_1 and d_2 are **token-equivalent** if there exists a concrete diagram $\hat{d}$ that complies with both.

Isomorphism of abstract diagrams is an algebraic notion, while token-equivalence is a syntactic notion; however, the two notions are related by the following theorem.

Theorem 2 *Two abstract diagrams are isomorphic if they are token-equivalent.*

Proof: (*Sketch of proof*) Let $\hat{d}$ be a concrete diagram that complies with abstract diagrams d_1 and d_2 and let $\mu_1 : \hat{d} \to d_1$ and $\mu_2 : \hat{d} \to d_2$ be abstraction mappings. Then the mapping $\theta = \mu_2 {\mu_1}^{-1}$ is an isomorphism between d_1 and d_2. ∎

Theorem 3 *For each concrete diagram there is an abstract diagram to which it complies.*

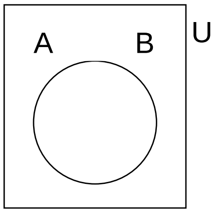

Fig. 12: An illegal diagram representing *A* = *B*.

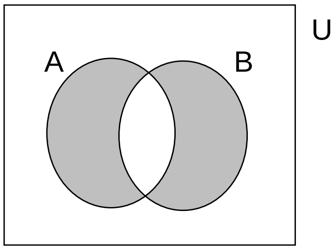

Fig. 13: A legal diagram representing *A* = *B*.

The proof is omitted. However, the converse is false. There are abstract diagrams that have no concrete instantiations. For example, the abstract diagram

$$d = \langle \mathcal{C}, \beta, \mathcal{Z}, \mathcal{Z}^*, \mathcal{L}, \ell, \mathcal{S}, \eta \rangle$$

where

$$\mathcal{C} = \{c_1, c_2\}$$

$$\mathcal{Z} = \{\emptyset, \{c_1, c_2\}\}$$

$$\mathcal{Z}^* = \emptyset$$

$$\mathcal{L} = \{A, B\}$$

$$\ell(c_1) = A, \ell(c_2) = B$$

$$\mathcal{S} = \emptyset$$

has no concrete instantiation. The only zones in the diagram are the zone outside all the contours (which must occur in all diagrams) and the zone $\{c_1, c_2\}$ which is within both c_1 and c_2. The only way of representing this concretely is by equating the two contours as illustrated in Fig. 12; this is illegal under (i) of the definition of a concrete diagram. Of course, the semantic information in this diagram can be represented as a concrete spider diagram, for instance, as in Fig. 13, but this diagram does not comply with d.

Two concrete diagrams

$$\hat{d}_1 = \langle \hat{\mathcal{C}}_1, \hat{\beta}_1, \hat{\mathcal{Z}}_1, \hat{\mathcal{Z}}_1^*, \hat{\mathcal{L}}_1, \hat{\ell}_1, \hat{\mathcal{S}}_1, \hat{\eta}_1 \rangle$$

and

$$\hat{d}_2 = \langle \hat{\mathcal{C}}_2, \hat{\beta}_2, \hat{\mathcal{Z}}_2, \hat{\mathcal{Z}}_2^*, \hat{\mathcal{L}}_2, \hat{\ell}_2, \hat{\mathcal{S}}_2, \hat{\eta}_2 \rangle$$

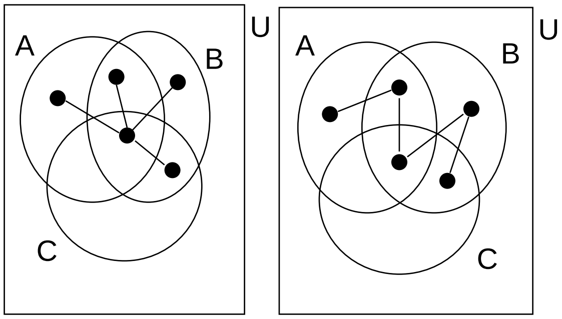

Fig. 14. Diagrammatically-equivalent diagrams with non-isomorphic spiders.

are **diagrammatically-equivalent** if there is a mapping $\phi : \hat{d}_1 \to \hat{d}_2$ that comprises a homeomorphism $\phi : \mathbf{R}^2 \to \mathbf{R}^2$ which induces bijections $\phi : \hat{\mathcal{C}}_1 \to \hat{\mathcal{C}}_2$, $\phi : \hat{\mathcal{Z}}_1 \to \hat{\mathcal{Z}}_2$, $\phi : F(\hat{\mathcal{S}}_1) \to F(\hat{\mathcal{S}}_2)$, where $F(\hat{\mathcal{S}}) \subseteq \mathbf{R}^2$ is the set of spiders feet in $\hat{d}$, and a bijection $\phi : \hat{\mathcal{L}}_1 \to \hat{\mathcal{L}}_2$. These mappings satisfy the following conditions:

1. $\forall \hat{c} \in \hat{\mathcal{C}}_1 \bullet \hat{\ell}_2(\phi(\hat{c})) = \phi(\hat{\ell}_1(\hat{c}))$.
2. $\forall \hat{z} \in \hat{\mathcal{Z}}_1 \bullet \hat{z} \in \hat{\mathcal{Z}}_1^* \Leftrightarrow \phi(\hat{z}) \in \hat{\mathcal{Z}}_2^*$.
3. $\forall \hat{f}, \hat{f}' \in F(\hat{\mathcal{S}}_1) \bullet \hat{f}$ and $\hat{f}'$ are feet of the same spider in $\hat{\mathcal{S}}_1$ if and only if $\phi(\hat{f})$ and $\phi(\hat{f}')$ are feet of the same spider in $\hat{\mathcal{S}}_2$.
4. $\forall \hat{f} \in F(\hat{\mathcal{S}}_1) \forall \hat{z} \in \hat{\mathcal{Z}}_1 \bullet \hat{f} \in \hat{\mathcal{Z}} \Leftrightarrow \phi(\hat{f}) \in \phi(\hat{z})$.

We only need to map $F(\hat{\mathcal{S}})$ because each spider is a tree and we only need to know the zones it touches, not its graph-theoretic isomorphism class. This is illustrated by the two diagrammatically equivalent spider diagrams in Fig. 14 in which the spiders are not graph-theoretically isomorphic, but do have equivalent habitats. This example also shows why the homeomorphism $\phi : \mathbf{R}^2 \to \mathbf{R}^2$ need not extend to spiders. The habitat of spider $\hat{s}$ is given by

$$\hat{\eta}(\hat{s}) = \{\hat{z} \in \hat{\mathcal{Z}} | \exists \hat{f} \in F(\hat{s}) \bullet \hat{f} \in \hat{z}\}.$$

Two concrete diagrams $\hat{d}_1$ and $\hat{d}_2$ are **type-equivalent** if there exists an abstract diagram d of which each is an instantiation.

Diagrammatic equivalence of concrete diagrams is a topological notion, while type-equivalence is a syntactic notion; however, the two notions are equivalent.

Theorem 4 *Two concrete diagrams are diagrammatically equivalent if and only if they are type-equivalent.*

5 Utility of two-tiered syntax

In this section we discuss the use of two-tiered syntax in diagrammatic reasoning and in the development of software tools to support such reasoning.

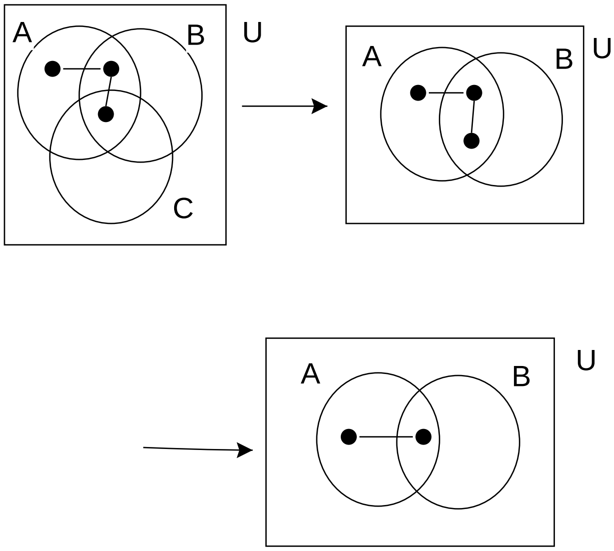

Fig. 15: Erasing a contour.

5.1 Diagrammatic reasoning

Diagrammatic reasoning is carried out by transforming diagrams, just as we manipulate sentences in order to make inferences in symbolic systems. As we discussed in §3, in the case of symbolic systems, it does not matter whether we mean to manipulate sentence-types or sentence-tokens, since making a type-token distinction does not have important consequences. On the other hand, in the case of diagrammatic systems, we need to make it clear whether transformation rules are being applied to diagram-tokens or to diagram-types.

It is natural to think that diagrammatic reasoning occurs at the token level, since what the user actually manipulates are concrete diagram-tokens not abstract diagram-types. However, we present several cases to illustrate that the relationship between type-syntax and token-syntax can be used to our advantage in carrying out the diagrammatic reasoning process. Thus, we argue that diagrammatic reasoning rules can be stated in terms of token-syntax but formalized and proved valid using type-syntax.

As a simple case, consider again the diagrams in Fig. 11. For example, we would like to allow the user to redraw the second diagram as a copy of the first diagram, or vice versa. The *copy rule* or the *reiteration rule* can be stated at the token-level, but with the help of the type-equivalence relation which is defined in the previous section.

Copy Rule: We may transform a concrete diagram $\hat{d}$ to another concrete diagram $\hat{d}'$ if and only if $\hat{d}$ and $\hat{d}'$ are type-equivalent.

A more interesting case is when we consider the reasoning rule, *contour erasure*. Erasing a contour can cause syntactic difficulties at the token level. In Fig. 14 erasing contour C results in two of the feet of the spider residing in the same zone. This is a not a well-formed diagram. To ensure that the resulting diagram is well-formed, the two feet in the same zone must be replaced with a single foot. A similar problem occurs with shading.

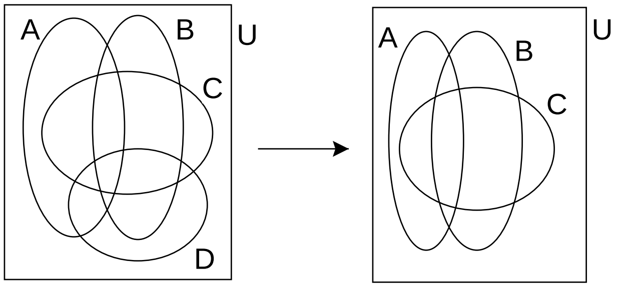

Fig. 16: Problem with erasing a contour.

The reasoning rule can be formulated at the token-syntax level in such a way that these difficulties are overcome:

Contour Erasure Rule (*token-level*): Let $\hat{d}$ be a concrete diagram with at least one contour and let $\hat{d}'$ be the diagram obtained from $\hat{d}$ after erasing a contour as follows:

1. any shading remaining in only a part of a zone is also erased;

2. if a spider has feet in two zones which combine to form a single zone with the erasure of the contour, then these feet are replaced with a single foot connected to the rest of the spider.

Then $\hat{d}$ can be replaced by $\hat{d}'$.

However, a more serious problem can occur. In Fig. 16 removing a contour results in a non-well-formed diagram because we get disconnected zones in the resulting diagram. This problem was first noticed by Scotto [21] as a flaw in Shin's Venn I system [22]. His solution involves combining the original diagram with the diagram composed just of a boundary contour. From a construction proved by More [18], we can produce a Venn diagram with at least one contour whose erasure would result in a well-formed diagram, this solution permits him to rearrange the contours so that the contour to be removed is this well-behaved contour.

We can offer a different solution. An alternative way of formulating the contour erasure rule is to consider an abstraction d of the concrete diagram $\hat{d}$ and to obtain the abstract diagram d' by removing a contour from d. Then $\hat{d}$ can be replaced by any instantiation of d'. The resulting diagram will be well-formed. The rule at the type level can be stated as follows.

Contour Erasure Rule (*type-level*): Let

$$d = \langle \mathcal{C}, \beta, \mathcal{Z}, \mathcal{Z}^*, \mathcal{L}, \ell, \mathcal{S}, \eta \rangle$$

be an abstract diagram with $c \in \mathcal{C}$ and let

$$d' = \langle \mathcal{C}', \beta', \mathcal{Z}', \mathcal{Z}'^*, \mathcal{L}', \ell', \mathcal{S}', \eta' \rangle$$

be the abstract diagram defined by:

1. $\mathcal{C}' = \mathcal{C} - \{c\}$ and $\beta' = \beta$.
2. There exists a surjection $\sigma : \mathcal{Z} \rightarrow \mathcal{Z}'$ defined by $\sigma(z) = z - \{c\}$. Furthermore, $z' \in \mathcal{Z}'^* \Leftrightarrow \forall z \in Z \bullet \sigma(z) = z' \Rightarrow z \in \mathcal{Z}^*$.
3. $\mathcal{L}' = \mathcal{L} - \{\ell(c)\}$ and $\ell' = \ell - \{c \mapsto \ell(c)\}$.
4. There is a bijection $\sigma : \mathcal{S} \rightarrow \mathcal{S}'$ satisfying $\eta'(\sigma(s)) = \sigma(\eta(s))$ (where we are using the natural extension of σ to regions).

Then d can be replaced by d'.

Any token-based solution is problematic in that complicated details of concrete syntax have to be considered, resulting in some very arcane conditions within the rule. The solution that we propose has the advantage of always producing a well-formed diagram in a natural way, because it is an instantiation of a diagram-type which has the required properties.

We suggest that this process extends to all diagrammatic reasoning rules. Therefore, we argue that diagrammatic reasoning takes place at the type-level of the system, even though we are operating on diagram-tokens. This way, we may take advantage of different kinds of flexibility each syntax-level has. At the same time, the mechanisms we developed in the previous section – instantiation, abstraction, type-equivalence, and token-equivalence – provide us with a guide to how to communicate between these two different levels. The following commutative diagram illustrates a general traffic rule: to transform concrete diagram $\hat{d}_1$ under a diagrammatic reasoning rule, we can transform an abstraction d_1 of $\hat{d}_1$ into abstract diagram d_2 and then any instantiation $\hat{d}_2$ of d_2 is the required transformation of $\hat{d}_1$.

$$\begin{tikzcd}
d_1 \arrow[r] & d_2 \arrow[d, "\zeta"] \\
\hat{d}_1 \arrow[u, "\mu"] \arrow[r] & \hat{d}_2
\end{tikzcd}$$

We will return to this discussion in the following section.

5.2 Software tools

For diagrammatic notations to be used on a large scale in the software development process, appropriate software tools must be developed. The process of diagrammatic reasoning by hand is difficult; however, with automated support the process becomes much easier and potentially very valuable. Our two-level syntax will play an important role in developing efficient software tools to aid the diagrammatic reasoning process. We claim that each level of ontology has its own merits, and moreover, that the close relationship between them will provide us with a more natural way to implement a diagrammatic system.

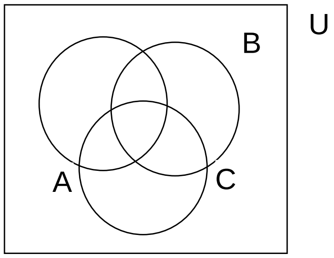

Fig. 17: Label ambiguity.

Note that the formalization of token-syntax we have given is a mapping to the plane, $\mathbf{R}^2$, and that this mapping itself is an abstraction. Hence, the formalization of token-syntax properly depends on the medium of the instantiation. For a computer instantiation, this would most likely be in terms of pixels. Because of the multiplicity of possible forms of instantiation, it is vital that we have a macro-level of syntax, i.e., the type-syntax, which provides us not only with the basic definition of a diagram but with the relation among different forms of instantiation.

In a concrete instantiation of a diagram presented on a sheet of paper, say, it can be difficult to determine an algorithm that gives the proper label for a contour. Is it the label closest to the contour? In Fig. 17, this is certainly not the case; on a global level the labelling can be disambiguated (by a human), but at a local level there are possible ambiguities (label B is not the closest label to any contour). However, in a computer instantiation of a diagram, it is easy to imagine an interactive process in which one clicks on a contour and its label is highlighted.

More importantly, in a given system, the communication established between two levels of syntax (illustrated at the end of the previous subsection) will help us to implement diagrammatic reasoning rules in an efficient way. Continuing the discussion of the contour erasure rule from the last section, it is easy to imagine a computer system in which the most appropriate instantiation of a diagram with a contour erased is produced automatically. In this case, the algorithm for this process must rely on the type-syntax of the diagram. At the same time, our abstraction and instantiation functions do an important part of the work so that a diagram-token on a computer monitor is transformed to another diagram-token.

6 Summary and further work

We have focused on a subset of the spider diagram notation and defined the type-syntax and token-syntax of the notation and the mappings between them. We have defined type-equivalent and token-equivalent diagrams and discussed the ontological issues involved. We have shown that it is necessary to consider both forms of syntax in developing diagrammatic inference systems and in their software implementation.

The general aim of this work is to provide the necessary mathematical underpinning for the development of software tools to aid reasoning with diagrams. In particular, we aim to develop the tools that will enable diagrammatic reasoning to become part of the software development process. In order for this to happen we need to be able to develop and implement an algorithm that takes a diagram-type and instantiates it as a diagram-token. Work is already underway on this [2].

Acknowledgements We would like to thank Jean Flower for comments on earlier drafts of this paper. We acknowledge Yan Sorkin, the creator of the *CDEditor* [7], the automatic tool that was used to generate most of the diagrams presented here.

References

[1] L. Euler. *Lettres a Une Princesse d'Allemagne*, volume 2. 1761. Letters No. 102–108.

[2] J. Flower. *Generating Constraint Diagrams*. MSc dissertation, University of Brighton, 2000.

[3] J. Gil, J. Howse, S. Kent. Formalising Spider Diagrams, *Proc. IEEE Symposium on Visual Languages (VL99)*, Tokyo, Sept 1999. IEEE Computer Society Press, 130-137.

[4] J. Gil, J. Howse, S. Kent, J. Taylor. Projections in Venn-Euler diagrams. *Proc. IEEE Symposium on Visual Languages* (VL2000), Seattle, Sept 2000. IEEE Computer Society Press, 119-126.

[5] J. Gil, J. Howse, E. Tulchinsky. Positive semantics of projections in Venn-Euler diagrams. *Proc. Diagrams 2000*, Edinburgh, Sept 2000. LNAI 1889, Springer-Verlag, 7-25.

[6] J. Gil, J. Howse, E. Tulchinsky. Positive semantics of projections. Accepted for the Journal of Visual Languages and Computing. To appear, 2001.

[7] J. Gil, Y. Sorkin. Ensuring Constraint Diagram Consistency: the *CDEditor* user-friendly approach. Manuscript available from second author. A copy of the editor is available at http::/www.geocities.com/ysorkin/cdeditor/. Apr. 2001.

[8] N. Goodman. *Languages of Art: An approach to a theory of symbols*. Hackett Publishing Company, INC. 1976.

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

[10] J. Howse, F. Molina, J. Taylor, S. Kent. Reasoning with Spider Diagrams. *Proc. IEEE Symposium on Visual Languages* (VL99), Tokyo, Sept 1999. IEEE Computer Society Press, 138-147.

[11] J. Howse, F. Molina, J. Taylor. SD2: A sound and complete diagrammatic reasoning system. *Proc. Artificial Intelligence and Soft Computing* (ASC 2000), Banff, July 2000. 402-408.

[12] J. Howse, F. Molina, J. Taylor. On the completeness and expressiveness of spider diagram systems. *Proc. Diagrams 2000*, Edinburgh, Sept 2000. LNAI 1889, Springer-Verlag, 26-41.

[13] J. Howse, F. Molina, J. Taylor. A sound and complete diagrammatic reasoning system. *Proceedings of IEEE Symposium on Visual Languages* (VL2000), Seattle, Sept 2000. IEEE Computer Society Press, 127-136.

[14] J. Howse, F. Molina, J. Taylor, S. Kent, J. Gil. Spider Diagrams: A Diagrammatic Reasoning System. Accepted for the Journal of Visual Languages and Computing. To appear, 2001.

[15] S. Kent. Constraint diagrams: Visualising invariants in object oriented models. *In Proceedings of OOPSLA97*, ACM SIGPLAN Notices 32, 1997.

[16] R. Lull. *Ars Magma*. Lyons, 1517.

[17] F. Molina. Reasoning with extended Venn-Peirce diagrammatic Systems. PhD Thesis, University of Brighton, 2001.

[18] T. More. On the construction of Venn diagrams. Journal of Symbolic Logic, 24, 303-304, 1959.

[19] Object Management Group. Unified Modeling Language Specification, Version 1.3. Available from www.omg.org.

[20] C. Peirce. *Collected Papers* Vol. 4. Harvard Univ. Press, 1933.

[21] P. Scotto di Luzio. Patching up a logic of Venn diagrams. Selected papers from the sixth CSLI Workshop on Logic, Language and Computation. CSLI Publications, Stanford, 2000.

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

[23] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag.*, 1880. 123.

[24] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.