On Diagram Tokens and Types

John Howse[1], Fernando Molina[1], Sun-Joo Shin[2], and John Taylor[1]

[1] School of Computing & Mathematical Sciences
University of Brighton, Brighton, UK
{John.Howse, F.Molina, John.Taylor}@brighton.ac.uk
http://www.it.bton.ac.uk/research/vmg/VisualModellingGroup.html

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

Abstract. Rejecting the temptation to make up a list of necessary and sufficient conditions for diagrammatic and sentential systems, we present an important distinction which arises from sentential and diagrammatic features of systems. Importantly, the distinction we will explore in the paper lies at a meta-level. That is, we argue for a major difference in meta-theory between diagrammatic and sentential systems, by showing the necessity of a more fine-grained syntax for a diagrammatic system than for a sentential system. Unlike with sentential systems, a diagrammatic system requires two levels of syntax—token and type. 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, the domains of type-syntax and token-syntax are distinct from each other. Euler diagrams are chosen as a case study to illustrate the following major points of the paper: (i) What kinds of diagrammatic features (as opposed to sentential features) require two different levels of syntax? (ii) What is the relation between these two levels of syntax? (iii) What is the advantage of having a two-tiered syntax?

1 Introduction

Many would suppose that there are fundamental differences between linguistic and diagrammatic systems. This assumption justifies a well-accepted classification among existing systems, for example, to call Euler or Venn systems diagrammatic and first-order languages linguistic. However, pinning down what those differences are has turned out to be a daunting task. The difficulty of the task has led some researchers in the area to a skepticism about any demarcation of different types of representation systems.

We find this skepticism vital as a guard against coming up with a quick and easy recipe which would easily ignore important theoretical issues involved in the nature of representation systems. On the other hand, we respect our on-going practice based on the intuition that there are different kinds of representation.

Recognizing merit in each position, we agree with many researchers that a relation between linguistic and diagrammatic systems should be understood to form a continuum.

Hence, we do not think that it is desirable to seek necessary and sufficient conditions to differentiate various kinds of systems. Instead, the issue of differences among systems should be refocused on clarifying linguistic or diagrammatic *elements* in any given representation system. That is, rather than making a comparison among different systems, we need to make a contrast among different features of a system. Depending on which features become salient in a system, it is called either diagrammatic or linguistic. Hence, strictly speaking, we are talking about *saliently* diagrammatic or *saliently* linguistic when we call a system either diagrammatic or linguistic. Again, we do not claim that we can or should complete a list of diagrammatic or linguistic features of a system, but the longer the list is, the more helpful it becomes.

How, then, can we produce a more comprehensive list. Conducting case studies on various systems is one of the most natural ways to achieve this goal. Exploring our intuitive ideas about various existing systems, this paper argues that it is necessary that a meta-theory of diagrammatic systems has different features from a meta-theory of linguistic systems.

Section 2 starts with the observation that a type-token distinction becomes a more important issue in a diagrammatic system than in a linguistic system. After identifying the main source of this discrepancy in diagrammatic and linguistic elements, we claim that a more fine-grained syntax is needed for diagrammatic representation. In §3, we develop a two-tiered syntax for the Euler system to sharpen our intuition underlying the type-token distinction in the system. With the formalism of §3 in hand, §4 illustrates the utility of two-level syntax, which leads us to the advantages of diagrammatic systems for certain purposes.

2 Tokens and types

The type-token issue is far from being settled in philosophy. However, there are important assumptions related to the type-token distinction that symbolic logic accepts without much controversy. Consider the following two pairs of symbolic sentences:

1. $(A_3 \wedge A_4)$
2. $(\mathsf{A}_3 \wedge \mathsf{A}_4)$
3. $(A_3 \wedge A_4)$
4. $(A_4 \wedge A_3)$

We may easily make a distinction among different kinds of sameness and difference in the above pairs: sentences (1) and (2) are different tokens of the same sentence type, sentences (3) and (4) belong to different sentence types but with the same meaning. A meta-theory of symbolic logic takes care of these kinds of difference in the following way. A difference among tokens (e.g. sentences (1)

and (2)) is ignored at the level of meta-theory, so, syntactically both tokens are treated as the same. The difference between (3) and (4) is reflected at the syntactic level, but semantics guarantees that these two sentences have the same meaning.

These distinctions sound almost trivial, but it is not a trivial question to explore the source of the triviality. Thanks to the conventions of a writing system, we know that neither the size nor the font of a symbol makes a difference to telling whether one string is the same as another. This convention is so ingrained that we almost do not see a visual difference between (1) and (2) or, even if we do, we consider the difference in font and size trivial. Therefore, an ambiguity between sentence-token and sentence-type, if any, is negligible. This is why the syntax of symbolic logic is always a type-syntax, not a token-syntax. On the other hand, every such symbolic system relies on a linear order, which is also related to the conventions of a writing system. Hence, sentences (3) and (4) belong to different types and the meaning equivalence of the two sentences is obtained in the semantics by the definition of the meaning of '$\wedge$.'

It is extremely interesting to notice that two features of a sentential system (discussed above) have determined two aspects of its meta-theory. It is well accepted that in a linguistic system neither size nor font of a symbol is a representing fact. Let us call this feature 'token-insensitivity.' The token-insensitive feature justifies a one-level syntax, that is, a type-syntax, of a system. The other feature of a linguistic system is a linear order or 'linearity.' This ingrained feature leads us to an easy agreement on different sentence-types for sentences (3) and (4). However, the price for this straightforward differentiation between (3) and (4) is that the system should have a commutative rule since semantic equivalence is not guaranteed any more for two sentences of different types.

We claim that both token-insensitivity and linearity are linguistic elements of a system and that a saliently diagrammatic system does not have either feature, that is, it is token-sensitive and non-linear. Accordingly, we argue that this difference is and should be reflected at the meta-level. We are making the following proposal: since a diagrammatic system is token-sensitive, it needs two-levels of syntax, that is, token-syntax and type-syntax. Also, we speculate that for a non-linear system a commutative rule is not needed. After illustrating token-sensitivity and non-linearity through examples, we will formalize our proposal and see that our speculation about commutativity is correct, at least for the system formalized.

Consider the three diagrams in Fig. 1. These are three different diagrams whose appearances are clearly distinguishable from one another. Visual differences in these three tokens are hard to ignore, but at the same time we know that these three diagrams are of the same kind in an important sense. In the case of the Venn system [13], the location of contours does not have 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 Euler diagrams in Fig. 2 are the same even though they represent the same fact. The semantics of the Euler

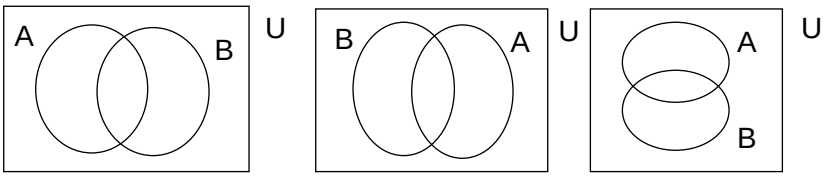

Fig. 1. Equivalent diagrams.

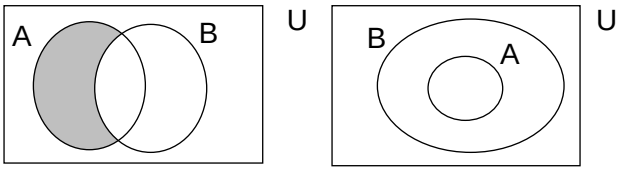

Fig. 2. Semantically-equivalent diagrams.

system tells us that these two diagrams represent the same fact (see §3 for more details of the Euler system we are illustrating). But, there is a clear syntactic distinction between the two diagrams: among other differences, there is shading in one diagram, but not in the other. That is, the difference and sameness in the case of Fig. 2 reflects an intuitive distinction between syntactic and semantic equivalences.

An interesting task is how to capture our intuition about the diagrams in Fig. 1: they are different from one another in one sense, but the same in another sense. First, we would like to decide whether we want to treat the sameness among the three diagrams at a syntactic level or at a semantic level. Again, using our discussions of a symbolic language, we may formulate the question in the following way: are the differences in Fig. 1 similar to the differences between sentences (1) and (2) or the differences between sentences (3) and (4)? We will show that neither of the options is satisfactory.

Suppose that we make no distinction between Fig. 1 and Fig. 2 and conclude that the diagrams in Fig. 1 are syntactically different from one another but semantically the same. That is, the differences among the diagrams in Fig. 1 are treated like the differences between sentences (3) and (4) in the case of symbolic logic. A problem with this approach is that the syntax would become so fine-grained that almost all diagrams would be syntactically different. Hence any equivalence among diagrams is postponed to a semantic level and we would require transformation rules to allow us to commute from one diagram to another in Fig. 1. These rules would be difficult to describe and would complicate the system so much that it would become unusable.

Suppose that we take the other position. The relation among the diagrams in Fig. 1 is just like the relation between sentences (1) and (2). They are syntactically equivalent to one another, period. A problem with this explanation

is that the syntax would become too coarse to accommodate differences among obviously present visual properties in diagrams. Interestingly, the reader would more easily notice visual differences among the three tokens of diagrams in Fig. 1 than the visual differences between sentences (1) and (2).

We claim that the discrepancy between syntactically equivalent diagrams and syntactically equivalent sentences is directly related to a difference between diagrammatic and linguistic elements of a system: while a saliently linguistic system adopts conventions of a writing system, a saliently diagrammatic system does not. Hence, there is no convention to rely on to decide that the position of contours is not a representing fact in a given system. It is perfectly possible to imagine a diagrammatic system in which the position of a syntactic object is a representing fact, for example, in many maps, A is to the left of B represents the fact that A is to the west of B. On the other hand, if the size or the font of a syntactic object becomes a representing fact, then we intuitively think that the system has a diagrammatic element.

We present a middle ground solution to the dilemma that these two alternatives present to us. If the syntax of a diagrammatic system is too fine-grained, then the diagrams in Fig. 1 are all syntactically different. But, if the syntax is not fine-grained enough, then we would have to accept a syntax that does not reflect non-trivial visual features of a system. We revive a time-honored distinction between type and token [4, 10] and suggest a two-tiered syntax for diagrammatic systems. Hence, the diagrams in Fig. 1 are *different tokens*, but of the *same type*. But, in the case of diagrams, differences in tokens are not negligible unlike with differences in sentence-tokens. 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 syntactic 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 consists 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. 1 are *different*, while the other level of syntax, i.e. type-syntax, lets us say that these diagrams are the *same*.

Thus, our solution seems to incorporate the intuition behind differences and equivalence among diagrams in Fig. 1. However, if the only advantage about making a distinction between type-syntax and token-syntax is to comply with our intuition and the result is to make a more complicated machinery for a system, one might wonder about the utility of the two-level syntax. After presenting a case study of the two-level syntax of the Euler system in the next section, we will return to this issue in §4.

3 Type-syntax and token-syntax of Euler diagrams

Euler diagrams [1] illustrate relations between sets. This notation uses topological properties of enclosure, exclusion and intersection to represent the set-theoretic notions of subset, disjoint sets, and intersection, respectively. In this

paper we have added shading to the notation to increase its expressiveness. Euler diagrams form the basis of more expressive diagrammatic notations such as Higraphs [7], constraint diagrams [3] and some of the notations of UML [9]. This paper is mainly concerned with diagrammatic notations based on Euler diagrams; this is a substantial and important collection in its own right as the above list indicates.

We now give a concise informal description of Euler diagrams. A *contour* is a simple closed plane curve. A *boundary rectangle* properly contains all other contours. Each contour has a unique label. 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 and 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. Every region is a union of zones. A region is *shaded* if each of its component zones is shaded. The set denoted by a shaded region is empty. In Fig. 2 the zone within A, but outside B is missing from the diagram; the set denoted by such a "missing" zone is also empty. An Euler diagram containing all possible zones is called a Venn diagram.

The **type-syntax** (or *abstract* syntax) of an Euler 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 screen, etc) that *complies* with the abstract definition; **token-syntax** describes instantiations. In this section we give the type-syntax of 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.

3.1 Type-syntax

An *abstract Euler diagram* is a tuple $d = \langle \mathcal{C}, U, \mathcal{Z}, \mathcal{Z}^* \rangle$ whose components are defined as follows:

1. $\mathcal{C}$ is a finite set whose members are called *contour labels*. The element U, which is not a member of $\mathcal{C}$, is called the *boundary rectangle label*.
2. 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.

At the abstract level, we identify the concepts of a contour and its label. A zone is defined by the contours that contain it and is thus represented as a set of contours. A region is just a non-empty set of zones.

3.2 Token-syntax

A *concrete Euler diagram* $\hat{d}$ is a tuple $\hat{d} = \langle \hat{\mathcal{C}}, \hat{\beta}, \hat{\mathcal{Z}}, \hat{\mathcal{Z}}^*, \hat{\mathcal{L}}, \hat{\ell} \rangle$ whose components are defined as follows:

1. $\hat{\mathcal{C}}$ 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 $\hat{\mathcal{C}}$ has the following properties:
   (a) Contours intersect transversely (i.e., locally, near the point of intersection, the diagram is a cross).
   (b) Each contour intersects with every other contour an even number of times; this, of course, could be zero times.
   (c) No two contours have a point in common without crossing at that point.
   (d) Each component of $\mathbf{R}^2 - \bigcup_{\hat{c}\in\hat{\mathcal{C}}} \hat{c}$ 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: $\bigcap_{\hat{c}\in X} \iota(\hat{c}) \cap \bigcap_{\hat{c}\in\hat{\mathcal{C}}-X} \varepsilon(\hat{c})$.
2. 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.
3. $\hat{\mathcal{L}}$ is the set of contour labels. The bijection $\hat{\ell} : \hat{\mathcal{C}} \cup \{\hat{\beta}\} \to \hat{\mathcal{L}}$ returns the label of a contour or the boundary rectangle.

Hammer [6] 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 left hand diagram in Fig. 3 in which two contours touch but do not cross is not well-formed by 1(c). Similarly, we ban diagrams with disconnected zones (1(d)), an example of which is given in the middle diagram of Fig. 3, which is attempting to represent $C \subseteq A \cup B$. 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. An alternative way to represent $C \subseteq A \cup B$ is given in the right hand diagram of Fig. 3 in which three contours intersect at a point, but all zones are connected; this is a legal construct.

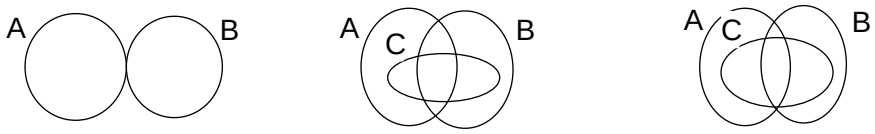

Fig. 3. Touching contours; a disconnected zone; a triple point.

3.3 Mappings between diagram types and tokens

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

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

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

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

1. $\forall c \in \mathcal{C} \bullet \hat{\ell}(\zeta(c)) = c$.
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. $\forall z \in \mathcal{Z} \bullet \zeta(z) \in \hat{\mathcal{Z}}^* \Leftrightarrow z \in \mathcal{Z}^*$.

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

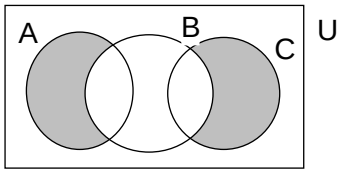

Fig. 4. A concrete Euler diagram.

The Euler diagram in Fig. 4 is an instantiation of the abstract diagram $d = \langle \mathcal{C}, U, \mathcal{Z}, \mathcal{Z}^* \rangle$ where $\mathcal{C} = \{A, B, C\}$, $\mathcal{Z} = \{\emptyset, \{A\}, \{A, B\}, \{B\}, \{B, C\}, \{C\}\}$, $\mathcal{Z}^* = \{\{A\}, \{C\}\}$.

We say that a concrete diagram *complies* with its abstraction. By forgetting the geometric information associated with a concrete diagram we have:

Theorem 1. *For each concrete diagram there is a unique abstract diagram to which it complies.*

However, there are abstract diagrams that have no concrete instantiations. For example, the abstract diagram $d = \langle \mathcal{C}, U, \mathcal{Z}, \mathcal{Z}^* \rangle$ where $\mathcal{C} = \{A, B\}$, $\mathcal{Z} = \{\emptyset, \{A, B\}\}$, $\mathcal{Z}^* = \emptyset$, 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 $\{A, B\}$ which is within both A and B. The only way of representing this

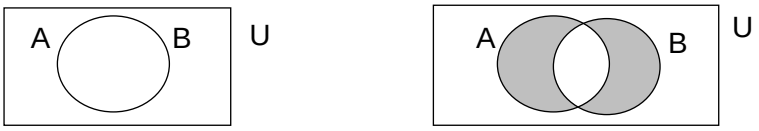

Fig. 5. Illegal and legal diagrams representing $A = B$.

concretely is by equating the two contours as illustrated in the left hand diagram of Fig. 5; this is illegal under 1(a),(c) of the definition of a concrete diagram. The semantic information in this diagram can be represented as a concrete diagram, for instance, as in the right hand diagram of Fig. 5, but this diagram does not comply with d.

3.4 Equivalent diagrams

Each abstract diagram has zero or many concrete instantiations. For example, the diagrams in Fig. 6 are both concrete representations of the abstract diagram $d_1 = \langle \mathcal{C}_1, U_1, \mathcal{Z}_1, \mathcal{Z}_1^* \rangle$ where $\mathcal{C}_1 = \{A, B\}$, $\mathcal{Z}_1 = \{\emptyset, \{A\}, \{A, B\}, \{B\}\}$, $\mathcal{Z}_1^* = \{\{A\}\}$.

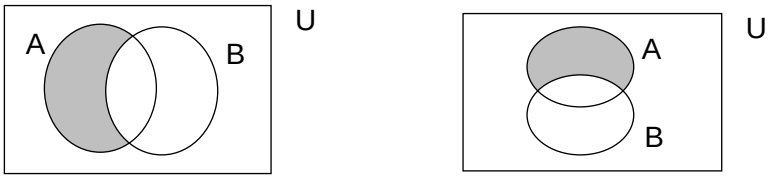

Fig. 6. Type-equivalent concrete diagrams.

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, i.e., if $\mu(\hat{d}_1) = d = \mu(\hat{d}_2)$. The two diagrams in Fig. 6 are type-equivalent. The diagrams in Fig. 1 in §2 are also type-equivalent. Type-equivalence allows us to say that two different looking diagram tokens are equivalent and is probably the most important concept in this paper; most of the points made in §4 on the utility of a two-tiered syntax rely on type-equivalence.

Two concrete diagrams $\hat{d}_1 = \langle \hat{\mathcal{C}}_1, \hat{\beta}_1, \hat{\mathcal{Z}}_1, \hat{\mathcal{Z}}_1^*, \hat{\mathcal{L}}_1, \hat{\ell}_1 \rangle$ and $\hat{d}_2 = \langle \hat{\mathcal{C}}_2, \hat{\beta}_2, \hat{\mathcal{Z}}_2, \hat{\mathcal{Z}}_2^*, \hat{\mathcal{L}}_2, \hat{\ell}_2 \rangle$ are **diagrammatically-equivalent** if $\hat{\mathcal{L}}_1 = \hat{\mathcal{L}}_2$ and 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 a homeomorphism $\phi : \bigcup_{\hat{c} \in \hat{\mathcal{C}}_1} \hat{c} \to \bigcup_{\hat{c} \in \hat{\mathcal{C}}_2} \hat{c}$ and bijections $\phi : \hat{\mathcal{C}}_1 \to \hat{\mathcal{C}}_2$, $\phi : \hat{\mathcal{Z}}_1 \to \hat{\mathcal{Z}}_2$. These mappings satisfy the following conditions:

1. $\forall \hat{c} \in \hat{\mathcal{C}}_1 \bullet \hat{\ell}_2(\phi(\hat{c})) = \hat{\ell}_1(\hat{c})$.

2. $\forall \hat{z} \in \hat{\mathcal{Z}}_1 \bullet \hat{z} \in \hat{\mathcal{Z}}_1^* \Leftrightarrow \phi(\hat{z}) \in \hat{\mathcal{Z}}_2^*$.

Diagrammatic equivalence of concrete diagrams is a topological notion, it captures our "geometric intuition" of equivalence, while type-equivalence is a syntactic notion; however, the two notions are related.

Theorem 2. *If two concrete diagrams are diagrammatically equivalent then they are type-equivalent.*

The proof is straightforward and is omitted.

Type-equivalence is a *label-preserving* equivalence. We now explore a *label-independent* equivalence. Consider the two diagrams in Fig. 7.

Fig. 7. Structurally equivalent concrete diagrams.

These two diagrams are different tokens of different types; however, if we remove the labels from the diagrams, they appear to be identical. We say that these two diagrams are *structurally equivalent* because their underlying diagrams (without the labels) are the "same."

Formally, two concrete diagrams $\hat{d}_1 = \langle \hat{\mathcal{C}}_1, \hat{\beta}_1, \hat{\mathcal{Z}}_1, \hat{\mathcal{Z}}_1^*, \hat{\mathcal{L}}_1, \hat{\ell}_1 \rangle$ and $\hat{d}_2 = \langle \hat{\mathcal{C}}_2, \hat{\beta}_2, \hat{\mathcal{Z}}_2, \hat{\mathcal{Z}}_2^*, \hat{\mathcal{L}}_2, \hat{\ell}_2 \rangle$ are **structurally equivalent** if there is a mapping $\phi : \hat{d}_1 \rightarrow \hat{d}_2$ that comprises a homeomorphism $\phi : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ which induces a homeomorphism $\phi : \bigcup_{\hat{c} \in \hat{\mathcal{C}}_1} \hat{c} \rightarrow \bigcup_{\hat{c} \in \hat{\mathcal{C}}_2} \hat{c}$ and bijections $\phi : \hat{\mathcal{C}}_1 \rightarrow \hat{\mathcal{C}}_2$, $\phi : \hat{\mathcal{Z}}_1 \rightarrow \hat{\mathcal{Z}}_2$, such that $\forall \hat{z} \in \hat{\mathcal{Z}}_1 \bullet \hat{z} \in \hat{\mathcal{Z}}_1^* \Leftrightarrow \phi(\hat{z}) \in \hat{\mathcal{Z}}_2^*$.

Now we consider equivalence of abstract diagrams. Abstract diagrams are many-sorted algebras, so the natural concept of equivalence is that of isomorphism.

Two abstract diagrams $d_1 = \langle \mathcal{C}_1, U_1, \mathcal{Z}_1, \mathcal{Z}_1^* \rangle$ and $d_2 = \langle \mathcal{C}_2, U_2, \mathcal{Z}_2, \mathcal{Z}_2^* \rangle$ are **isomorphic** if there is a mapping $\theta : d_1 \rightarrow d_2$ such that component mappings $\theta : \mathcal{C}_1 \rightarrow \mathcal{C}_2$, $\theta : \mathcal{Z}_1 \rightarrow \mathcal{Z}_2$, are each bijections and satisfy the following conditions:

1. $\forall c \in \mathcal{C}_1, \forall z \in \mathcal{Z}_1 \bullet c \in z \Leftrightarrow \theta(c) \in \theta(z)$ and $\theta(U_1) = U_2$.
2. $\forall z \in \mathcal{Z}_1 \bullet z \in \mathcal{Z}_1^* \Leftrightarrow \theta(z) \in \mathcal{Z}_2^*$.

The abstract diagrams of the two diagram tokens in Fig. 7 are $d_1 = \langle \mathcal{C}_1, U_1, \mathcal{Z}_1, \mathcal{Z}_1^* \rangle$ where $\mathcal{C}_1 = \{A, B\}$, $\mathcal{Z}_1 = \{\emptyset, \{A\}, \{A, B\}\}$, $\mathcal{Z}_1^* = \emptyset$ and $d_2 = \langle \mathcal{C}_2, U_2, \mathcal{Z}_2, \mathcal{Z}_2^* \rangle$ where $\mathcal{C}_2 = \{A, B\}$, $\mathcal{Z}_2 = \{\emptyset, \{B\}, \{A, B\}\}$, $\mathcal{Z}_1^* = \emptyset$. Then $\theta : d_1 \rightarrow d_2$, where $\theta(A) = B$ and $\theta(B) = A$ is an isomorphism.

Structural equivalence of tokens is a topological notion, while isomorphism of types is an algebraic notion; again the two notions are related.

Theorem 3. *If two concrete diagrams $\hat{d}_1$, $\hat{d}_2$ are structurally equivalent then their abstractions $\mu(\hat{d}_1)$ and $\mu(\hat{d}_2)$ are isomorphic.*

This is just the label-free version of Theorem 2. Structural equivalence of tokens (and hence isomorphism of their abstractions) is important for the classification of diagrams, which is itself important for software tool building and diagrammatic reasoning. However, if two tokens are structurally equivalent, they need not be semantically equivalent. This leads us to yet another level of equivalence of diagrams—*semantic equivalence*. If two tokens are type-equivalent, then they are semantically-equivalent. The two tokens in Fig. 2 are not type-equivalent, but they are semantically equivalent, while the two tokens in Fig. 7 are structurally equivalent, but not semantically equivalent.

4 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. Diagrammatic reasoning can take many forms. In this paper we are concerned with self-contained diagrammatic systems involving diagram manipulation.

4.1 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 §2, 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. 6. We would like to allow redrawing the second diagram as a copy of the first diagram, or vice versa. The *copy 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. 8 erasing contour C results in shading in part of a zone. This is not a well-formed

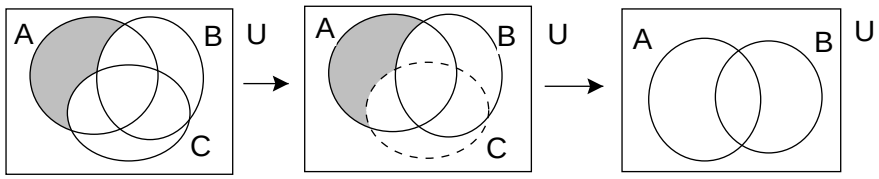

Fig. 8. Erasing a contour.

diagram. To ensure that the resulting diagram is well-formed, the shading must be removed. The reasoning rule can be formulated at the token-syntax level in such a way that this difficulty is 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 and erasing any shading remaining in only a part of a zone. Then $\hat{d}$ can be replaced by $\hat{d}'$.

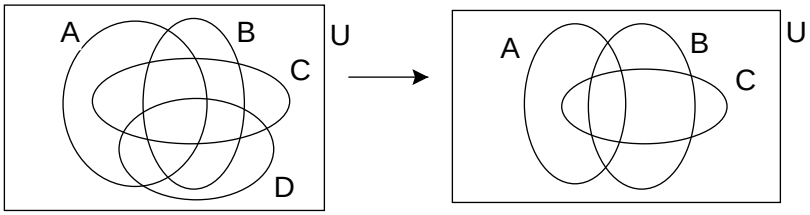

Fig. 9. Problem with erasing a contour.

However, a more serious problem can occur. In Fig. 9 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 [11] as a flaw in Shin's Venn I system [13]. His solution involves combining the original diagram with the diagram composed just of a boundary contour. From a construction proved by More [8], we can construct a Venn diagram with at least one contour whose erasure would result in a well-formed diagram. Scotto's solution rearranges 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 concrete diagram will be well-formed. The rule at the type level can be stated as follows.

Contour Erasure Rule (*type-level*): Let $d = \langle \mathcal{C}, U, \mathcal{Z}, \mathcal{Z}^* \rangle$ be an abstract diagram with $c \in \mathcal{C}$ and let $d' = \langle \mathcal{C}', U', \mathcal{Z}', \mathcal{Z}'^* \rangle$ be the abstract diagram defined by:

1. $\mathcal{C}' = \mathcal{C} - \{c\}$ and $U' = U$.
2. There exists a surjection $\sigma : \mathcal{Z} \to \mathcal{Z}'$ defined by $\sigma(z) = z - \{c\}$. Furthermore, $z' \in \mathcal{Z}'^* \Leftrightarrow \forall z \in Z \bullet \sigma(z) = z' \Rightarrow z \in \mathcal{Z}^*$.

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 diagrammatic manipulation rules in general. What we draw and manipulate are tokens of diagrams. In this sense, diagrammatic reasoning takes place at the token level, but the rules of these operations are stated at the type level, so in another sense diagrammatic reasoning takes place at the type level. 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 $\hat{d_1}$.

$$\begin{array}{ccc} d_1 & \longrightarrow & d_2 \\ {\scriptstyle\mu}\uparrow & & \downarrow{\scriptstyle\zeta} \\ \hat{d_1} & \longrightarrow & \hat{d_2} \end{array}$$

In §2, we have seen that the necessity of a type-token distinction arises from the token-sensitivity and non-linearity of a diagrammatic system. Based on the formalism presented in the above section, in this section we show that this necessary mechanism adds efficiency to the system.

The following two sentences are syntactically different but semantically equivalent.

$$\forall x(Px \to \neg Qx)$$
$$\forall x(Qx \to \neg Px)$$

If the semantics is correct, we are able to prove that the two sentences always get the same truth value under the same interpretation. If the system is complete, we have a way to manipulate one sentence to the other.

Let's compare this situation with Fig. 6. Our formalism in the previous section tells us that these two diagrams are type-equivalent. So, we do not need to

explore the semantics to see the semantic equivalence. The equivalence is taken care of at a syntactic level. Since they belong to the same type, the copy rule of the system simply allows the manipulation between these two tokens. When a type-token distinction is implemented in syntax, the syntactic operation between these two diagrams comes almost free. Shimojima [12] uses term 'free ride' to refer to an inference in which the conclusion seems to be read off almost automatically from the representation of premises. For further discussion, see [5]. Consider again the diagrams in Fig. 6. Type-equivalence in a syntactic level gives us a free pass for a commutation between these two diagrams without bringing in transformation rules (except the trivial copy rule). Since type-equivalence implies semantic equivalence, we get another free pass for the semantic relation between these two diagrams. Clearly, the relation between "$\exists x(Px \wedge Qx)$" and "$\exists x(Qx \wedge Px)$" involves a more elaborate story.

This efficacy is directly related to our speculation in §2, that is, that a non-linear system might not need the commutative rule. The speculation turns out to be correct, at least for the Euler system, but only when our intuition about a type/token distinction is formalized in type-syntax and token-syntax is the truth of the speculation proven. Interestingly, the 'free ride' commutativity in some diagrams can be problematic; see [3] for a discussion on such problems in the representation of quantifiers in constraint diagrams.

4.2 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.

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 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 screen is transformed to another diagram-token.

5 Summary and further work

We have discussed and formally defined a two-tiered-syntax—type and token—of a diagrammatic system. We have compared this to the situation in linguistic systems and claim that this is a distinguishing feature, at least for diagrammatic notations based on Euler diagrams; this is a substantial and important collection in its own right but it is our belief that our claim extends to other types of diagram. We have shown that it is necessary to consider both forms of syntax in developing "self-contained" 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 and the anonymous referees for their very helpful comments. Authors Howse and Taylor were partially supported by UK EPSRC grant GR/R63516.

References

1. L. Euler. *Lettres a Une Princesse d'Allemagne*, vol 2. 1761. Letters No. 102-108.
2. J. Flower, J. Howse. *Generating Euler Diagrams.* Accepted for Diagrams 2002.
3. J. Gil, J. Howse, S. Kent. Towards a formalization of constraint diagrams, *Proc Symp on Human-Centric Computing*, Stresa, Sept 2001. IEEE Press
4. N. Goodman. *Languages of Art: An approach to a theory of symbols.* Hackett Publishing Co, INC. 1976.
5. C. Gurr, J. Lee and K. Stenning. Theories of diagrammatic reasoning: Distinguishing component problems, in *Minds and Machines* 8, 533-557, 1998.
6. E. Hammer. *Logic and Visual Information.* CSLI Publications, 1995.
7. D. Harel. On visual formalisms. In J. Glasgow, N. H. Narayan, B. Chandrasekaran, eds, *Diagrammatic Reasoning*, 235-271. MIT Press, 1998.
8. T. More. On the construction of Venn diagrams. J Symb Logic, 24, 303-304, 1959.
9. OMG. UML Specification, Version 1.3. Available from www.omg.org.
10. C. Peirce. *Collected Papers* Vol. 4. Harvard Univ. Press, 1933.
11. P. Scotto di Luzio. Patching up a logic of Venn diagrams. Proc. 6th CSLI WS on Logic, Language and Computation. CSLI Publications, Stanford, 2000.
12. A. Shimojima. Operational constraints in diagrammatic reasoning, in J. Barwise, G. Allwein eds. *Logical Reasoning with Diagrams*. OUP, New York, 27-48, 1996.
13. S.-J. Shin. *The Logical Status of Diagrams.* CUP, 1994.
14. J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag.*, 1880. 123.