Towards a Formalization of Constraint Diagrams

Joseph (Yossi) Gil
Department of Computer Science
Technion—IIT, Haifa 32000, Israel
yogi@cs.technion.ac.il

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

Stuart Kent
Computing Laboratory
University of Kent, Canterbury, UK
S.J.H.Kent@ukc.ac.uk

Abstract

Geared to complement UML and to the specification of large software systems by non-mathematicians, constraint diagrams *are a visual language that generalizes the popular and intuitive Venn diagrams and Euler circles, and adds facilities for quantifying over elements and navigating relations. The language design emphasizes scalability and expressiveness while retaining intuitiveness. Spider diagrams form a subset of the notation, leaving out universal quantification and the ability to navigate relations. Spider diagrams have been given a formal definition. This paper extends that definition to encompass the constraint diagram notation. The formalization of constraint diagrams is nontrivial: it exposes subtleties concerned with the implicit ordering of symbols in the visual language, which were not evident before a formal definition of the language was attempted. This has led to an improved design of the language.*

Keywords Visual formalisms, software specification, formal methods.

1 Introduction

The uptake in the software industry of notations for designing systems visually has been accelerated with the standardization of the Unified Modeling Notation (UML) [13]. UML brings together a number of informal visual notations that have proven useful to some parts of industry. It has made little progress in integrating these notations, although formalization of UML notations, in various forms, is currently a hot research topic.

In this paper, we describe a notation, *constraint diagrams*, which was introduced in [10] as a visual technique intended to be used in conjunction with UML for object-oriented modelling. Constraint diagrams provide a diagrammatic notation for expressing constraints (invariants) that can only be expressed in UML using the Object Constraint Language (OCL) [17], essentially a textual, stylised form of first order predicate logic, which is part of the UML standard [13].

Constraint diagrams were developed to enhance the visualization of object structures and they build on class diagrams in UML. Whereas class diagrams are only able to show that there are relationships between certain kinds of object, constraint diagrams are able to visualize properties of those relationships and compositions of those relationships.

Constraint diagrams build on a long history of using diagrams to visualize logical or set-theoretical assertions. Circles or closed curves, which we call contours, have been in use for the representation of classical syllogisms since at least the Middle Ages [12]. 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.

The logician John Venn used contours to represent logical propositions [16]. 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 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 logician Charles Peirce augmented Venn diagrams by adding X-sequences as a means for denoting elements [14]. An X-Sequence connecting a number of “minimal regions” of a Venn diagram, indicates that their union is not empty. Full semantics and inference rules have only recently been developed for Venn-Peirce diagrams [15] and Euler circles [5]. Constraint diagrams bear a resemblance to Harel’s *higraphs* [6], the basis of *statecharts*, in that they both represent binary relations by using arrows between closed curves. However, the semantics of the two notations are rather different.

Spider diagrams [2] 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. They emerged from work on constraint diagrams and such is the similarity with Venn-Peirce and Euler diagrams that the methods used on the existing notations were adapted to give formal semantics and develop sound and complete inference rules for the more expressive spider diagrams. The syntax and informal semantics of spider diagrams are considered in §2. We then extend this, in §3, to develop the syntax and informal semantics of constraint diagrams. In the work on constraint diagrams it soon became apparent that the notation was far more sophisticated than it first seemed. Specifically we started to discover examples where, although there seemed to be an intuitive reading, it was not obvious how that reading was derived in any general or systematic way. These issues are discussed in §4. In §5 we develop the formal syntax and a partial semantics of constraint diagrams. Finally, in §6, we conclude the paper and discuss future research, including an idea that resolves most of the issues discussed in §4.

Constraint diagrams include the idea of *projections*, which remove clutter and focus attention in the diagram appropriately. For space reasons, we are unable to consider projections in this paper; for details see [3].

2 Spider diagrams

In this section we give a concise description of spider diagrams. For more details see [2]. A *contour* is a simple closed plane curve. A *boundary rectangle* properly contains all other contours, although we do not usually represent it in concrete spider or constraint 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 *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. There is a slight semantical difference between spiders with circles as feet and spiders with squares as feet. A spider whose feet are circles corresponds to existential quantification of an element within the set denoted by the habitat of the spider. A spider whose feet are squares represents a given element. A *Schrödinger* spider denotes a set whose size is either zero or one. Like Schrödinger's cat, one is not sure whether the element within the set exists or not. The feet of a Schrödinger spider are rendered as open circles. Fig. 1 contains examples of all these types of spider.

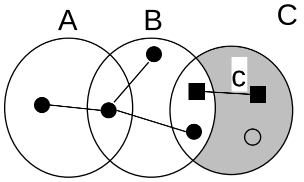

Fig. 1: Spider diagram.

Two distinct spiders denote distinct elements, unless they are joined by a *tie* or by a *strand*. A tie is a double, straight line (like an equals sign) connecting two feet, from different spiders, placed in the same zone. The *nest* of spiders s and t, written $\tau(s,t)$, is the set of those zones z having the property that the feet of s and t are connected by a tie in z. Two spiders which have a non-empty nest are referred to as *mates*. If both the elements denoted by spiders s and t are in the set denoted by the same zone in the nest of s and t, then s and t denote the same element. A *strand* is a wavy line connecting two feet, from different spiders, placed in the same zone. The *web* of spiders s and t, written $\zeta(s,t)$, is the union of zones z having the property that there is a sequence of spiders $s = s_0, s_1, \ldots, s_n = t$ such that, for $i = 0, \ldots, n-1, s_i$ and s_{i+1} are connected by a tie or by a strand in z. So $\tau(s,t)$ is a subregion of $\zeta(s,t)$. Two spiders with a non-empty web are referred to as *friends*. Two spiders s and t may (but not necessarily must) denote the same element if that element is in the set denoted by the web of s and t.

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 in the same unitary diagram can have the same label. The labelling of spiders is optional. The semantics of the diagram in Fig. 1 includes $A \cap C = \emptyset \wedge c \in C \wedge |C - B| \leq 2 \wedge \exists x \in A \cup B \bullet x \neq c$. The given spider in C is labelled c.

Sound and complete diagrammatic inference rules have been developed for several systems of spider diagrams [7,

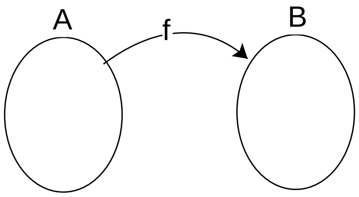

Fig. 2: Simple arrow.

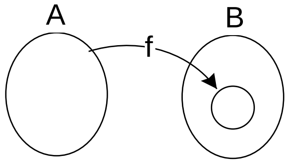

Fig. 3: A derived contour.

8, 9].

3 Syntax and informal semantics of constraint diagrams

Constraint diagrams are spider diagrams augmented by *arrows* and *universal spiders*. Arrows represent binary relations between sets and universal spiders denote universal quantification.

An arrow has a *label*, a *source* and a *target*. The source of an arrow can be a contour, a zone or a spider. The target of an arrow can be a contour, a zone or a spider; it is interpreted as the relational image of the set represented by the source under the relation represented by the arrow. We will use the standard object-oriented notation $x.r$ to represent textually the relational image of element x under relation r, i.e., $x.r = \{y : (x, y) \in r\}$; thus $x.r$ is the set of all elements related to x under relation r. The expression $x.r$ is a *navigation expression*, so called because we can *navigate* from x along the arrow r to the set $x.r$. The relational image of a set S is then given by $S.r = \bigcup_{x \in S} x.r$ (note that, in OCL, the navigation expression $S.r$ returns a bag; constraint diagrams are purely set-based). Let S and T be the sets represented by the source and target, respectively, of arrow r. Then the equation $S.r = T$ holds. In Fig. 2 the source of arrow f is contour A and its target is contour B. Its interpretation is $A.f = B$.

In Fig. 3, the target of arrow f is a *derived contour*. The interpretation is $A.f \subseteq B$. Derived contours are non-given contours appearing as the target of some arrow. The set denoted by a derived contour is defined by the navigation expression of its arrow (or arrows).

The source or target of an arrow can be a zone. In Fig. 4, the interpretation is $(A \cap B).f = (C - D)$. Both the source and target of the arrow in Fig. 5 are given spiders; the interpretation is $x.f = y$. The target of an arrow should be a set. However, the arrow is targeted on a given spider. This spider is treated semantically as a singleton set. We are implicitly allowing coercion between a singleton set and its element.

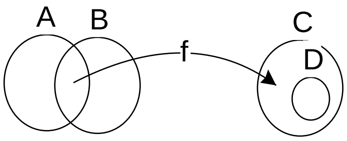

Fig. 4: Arrow with zone as source and zone as target.

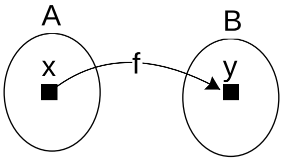

Fig. 5: Arrow with given spiders as source and target.

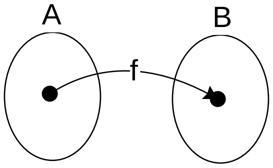

Fig. 6: Existential and derived spiders.

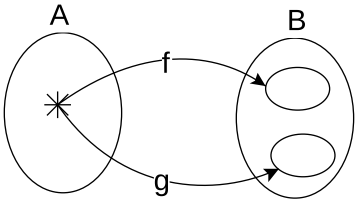

Fig. 7: Arrows with universal spider as source.

3.1. Quantification

The notation is able to express existential and universal quantification. The navigation expression in Fig. 6 is existentially quantified, as its source is an existential spider. Its interpretation is $\exists x \in A \bullet x.f \in B$. The target is a *derived* spider. This spider is treated as if it were a derived contour whose interpretation is a singleton set and, because it is a singleton set, we coerce the set into its element and hence write $x.f \in B$ rather than $x.f \subseteq B$.

The source of an arrow can also be a *universal spider*. The universal spider, whose feet are rendered as $*$s, ranges over all the elements of the set denoted by its habitat. In Fig. 7, the universal spider ranges over all elements of the set A. The interpretation of this diagram is that for each x in A, $x.f$ and $x.g$ are disjoint, i.e., $\forall x \in A \bullet x.f \cap x.g = \emptyset$.

A universal spider can only be the source of an arrow. No arrow can be targeted on a universal spider and a uni-

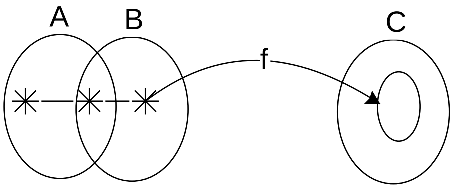

Fig. 8: Arrow with articulated universal spider as source.

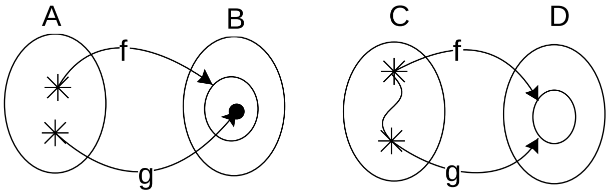

Fig. 9: Distinct and non-distinct universal spiders.

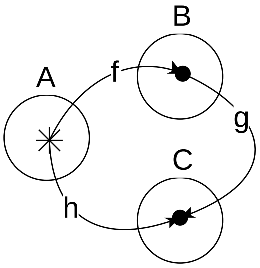

Fig. 10: Navigation.

versal spider that is not the source of an arrow cannot exist (it would have no meaning; this is why universal spiders do not occur in spider diagrams). A universal spider can be articulated (i.e., have more than one foot). In Fig. 8, the source of arrow f is an articulated universal spider. The arrow can be sourced on any part of the spider, that is, on any of the feet or on any of the legs. The interpretation of this is $\forall x \in A \cup B \bullet x.f \subseteq C$. The universal spiders in contour A of Fig. 9 obey the rule that spiders represent distinct elements unless connected by a strand or a tie. So the interpretation is $\forall x, y \in A \bullet x \neq y \Rightarrow y.g \in x.f$. The universal spiders in C of Fig. 9 are connected by a strand and therefore can represent the same elements; the interpretation is $\forall x, y \in C \bullet y.g = x.f$.

3.2. Constraint diagrams and OO modelling

So far, we have only considered single-arrow navigation expressions. In Fig. 10, there is a two-arrow navigation expression $\forall x \in A \bullet x.f.g$. The expression $x.f.g$ is interpreted as $(x.f).g$ which is equal to $\bigcup_{y \in x.f} y.g$; we take the union of the image sets under g of each element of the derived set $x.f$. Interestingly, if we take the expression $f.g$ to be the composition of f and g, then we have $(x.f).g = x.(f.g)$ and hence the expression $x.f.g$ is well-defined, albeit with a very overloaded *dot*; this, of course, is not the case in OCL, where $x.f.g$ delivers a bag. In Fig. 10, the set obtained by navigating from the universal spider in A via arrows f and g is the same as that obtained by navigating from it along h. Thus we have $\forall x \bullet x.f.g = x.h$. One of the key strengths of constraint diagrams is their ability to illustrate diagrammatically navigation expressions and the relationships between them.

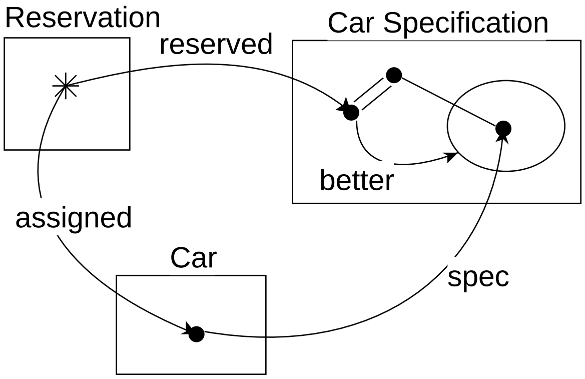

Fig. 11: A car-hire constraint.

When constraint diagrams are used in object-oriented modelling, we use rectangles for classes (or types) and round-cornered rectangles for states and we identify a state as a subset of a class. The constraint diagram in Fig. 11 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.

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

4 Issues

In the work on the development of constraint diagrams it became apparent that the notation was rather more sophisticated than we anticipated. Specifically, we started to discover examples where, although there seemed to be an intuitive reading, it was not obvious how that reading was derived in any general or systematic way. These examples generally involved quantification and interesting relationships between navigation expressions. For instance, consider the diagram in Fig. 10. The target of arrow f is a derived spider (a derived singleton set), which, in turn, is the source of arrow g; the target of g is also a derived spider. So, both the source and target of g depend on f; indeed, they are both parts of a universally quantified statement

$$\forall x \in A \bullet (x.f \in B \wedge (x.f).g \in C).$$

To consider the source and target of g outside the scope of this quantified statement would be meaningless; some arrows cannot be interpreted independently. To be precise, any arrow whose source is a derived contour or a derived

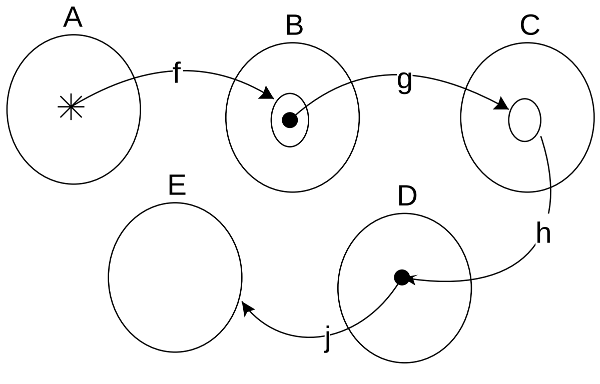

Fig. 12: An arrow sequence.

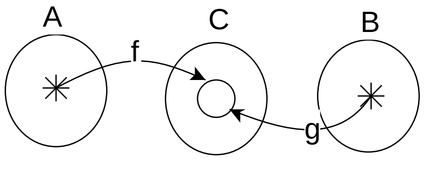

Fig. 13: Two universally quantified arrows with same target.

spider or is a universal or existential spider whose habitat is a derived contour is intrinsically linked to another arrow. Of course, in longer navigation expressions we can have arrows dependent upon many other arrows for their interpretation. In order to give semantics to such expressions, we have to consider whole sequences of arrows.

We define an *arrow sequence*, to be a sequence $a_1, \ldots, a_n$ of distinct arrows with the following properties. The source of a_1 is a quantified spider whose habitat is not derived. The source of each of the other spiders is a derived contour or spider or is a quantified spider whose habitat is a derived contour. The target of each of the arrows except (possibly) the last one (a_n) is derived and the target of a_i is equal to the source of a_{i+1} or is the habitat of the quantified spider that is the source of a_{i+1}, for $i \in 1..n-1$. An arrow sequence $a_1, \ldots, a_n$ in which the target of a_n is not derived or if it is derived it does not contain a quantified spider that is the source of another arrow is called a *complete arrow sequence*.

In Fig. 12, there is a complete arrow sequence f, g, h, j, where each arrow has a different kind of source. The arrow sequence is interpreted as $\forall x \in A \bullet (x.f \subseteq B \wedge \exists y \in x.f \bullet (y.g \subseteq C \wedge y.g.h \in D \wedge y.g.h.j = E))$. When obtaining the formal semantics of constraint diagrams, it is arrow sequences we need to consider rather than that of individual arrows. Of course, a single arrow can constitute a complete arrow sequence. Arrow sequences are given formal definitions and formal semantics in §5.

An interesting case occurs when two quantified arrows have the same derived target. This means that the target of each arrow is interpreted as the same set. Consider Fig. 13. There are two interpretations of this, depending on which of the spiders we start from: $\forall x \in A \bullet (x.f \subseteq C \wedge \forall y \in B \bullet (y.g \subseteq C \wedge x.f = y.g))$ and $\forall y \in B \bullet (y.g \subseteq C \wedge \forall x \in A \bullet (x.f \subseteq C \wedge x.f = y.g))$. The expression $x.f = y.g$ must be within the scope of both the universal quantifiers. However, the two interpretations are in fact equivalent, as they can be rewritten as

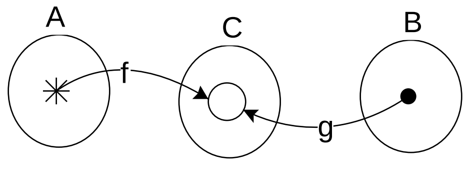

Fig. 14: Ordering of quantifiers.

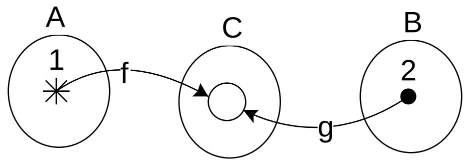

Fig. 15: Numbered quantifiers.

$$\forall x \in A \forall y \in B \bullet (x.f \subseteq C \wedge y.g \subseteq C \wedge x.f = y.g)$$

and the ordering of the two universal quantifiers is irrelevant. In fact, this expression constrains the relation f so that $\forall x_1, x_2 \in A \bullet x_1.f = x_2.f$ and similarly for g.

However, a problem arises when we consider a similar situation but with a universal quantifier and an existential quantifier. The ordering of universal and existential quantifiers is important. The diagram in Fig. 14 can have two very different interpretations, depending on which of the spiders we start from: $\forall x \in A \exists y \in B \bullet x.f = y.g$ and $\exists y \in B \forall x \in A \bullet x.f = y.g$.

There are a number of solutions to this type of problem. We could ban such expressions. It is not too difficult to construct a syntactic condition that would exclude such cases. This, of course, reduces the expressiveness of the notation.

We could have a default semantics which is always unique; for instance we could insist that universal quantifiers always had precedence over existential quantifiers (where there was any ambiguity). This would mean that the first interpretation held; however, it would then be impossible to express the second interpretation in the notation, again reducing the expressiveness of the notation.

A third possibility is to introduce more syntax into the notation. For instance, quantifiers could be numbered, or even coloured, to indicate the order in which they occur. In Fig. 15 precedence is given to the universal quantifier as it is numbered 1. The interpretation is the first one given above; with the numbering reversed, the second interpretation would hold. In an arrow sequence the order of the quantifiers is given by the order of the arrows, however, when two sequences interact, there could be a problem; the

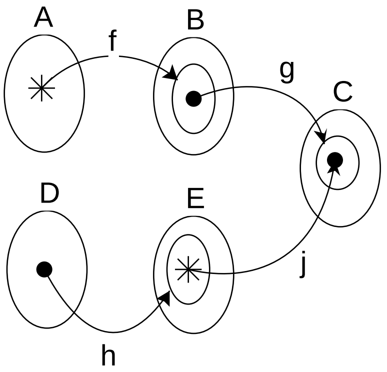

Fig. 16: Two interacting sequences.

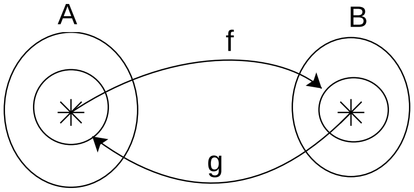

Fig. 17: Circularity.

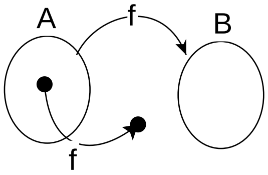

Fig. 18: Inconsistency.

simplest case of this is Fig. 14. If all the quantifiers in two interacting arrow sequences are universal or if all of them are existential, then there is an unambiguous interpretation; the simplest case of this is Fig. 13. In the case in which there is a mixture of types of quantifier, we could number the quantifiers to remove any ambiguity, but the ordering of the quantifiers in any arrow sequence must follow the ordering of the arrows. In Fig. 16, there are at least six different interpretations. The quantifier in A has precedence over the quantifier in B and the quantifier in D has precedence over the quantifier in E, apart from that any permutation of the quantifiers would be permitted.

A difficult issue is that of *circularity*. Consider Fig. 17. Each derived contour is defined in terms of the other derived contour. There is no arrow sequence in this diagram. It is very difficult to interpret this diagram without being told where to start. One interpretation might be to start navigating from the universal spider in A (we'll call it x), in which case it should range over all the elements in A because the derived contour containing it is defined in terms of an arrow from a universal spider (which we'll call y) which is contained in a derived contour which is defined in terms of x. So an interpretation is $\forall x \in A \bullet (x.f \subseteq B \wedge \forall y \in x.f \bullet (y.g \subseteq A \wedge x \in y.g))$. In this interpretation, the universal spider within B ranges over the elements of the derived set $x.f$, not over the whole of B. The symmetry of the expression is broken and it does matter where we start the interpretation, because, of course, by symmetry, there is a similar, but different, interpretation starting with the universal spider in B, i.e., $\forall y \in B \bullet (y.g \subseteq A \wedge \forall x \in y.g \bullet (x.f \subseteq B \wedge y \in x.f))$. We can, again, number the quantifiers to give an unambiguous reading, but this interpretation does require the scope of the universal spider in A to be something other than its habitat; this may well be counter-intuitive.

The obvious thing to do in such cases is to ban such expressions and for the moment this is what we will do. This does limit the expressiveness of the notation, but it prevents a possible ambiguity. We can banish these expressions syntactically by insisting that all arrows must be part of some complete arrow sequence.

In §6, we sketch the idea of *constraint trees*, which deals with many of the problems of expressibility and allows us to modularize the notation and to mix it with other notations. A full consideration of this idea is beyond the scope of this paper.

Finally in this section, we will consider the problem of inconsistency. For spider diagrams, we can prove that each unitary diagram is consistent, i.e., that it has a compliant model. The constraint diagram in Fig. 18 is inconsistent as $A.f = B$, but there exists $x \in A$ for which $x.f \notin B$. It may well be possible to syntactically exclude all such inconsistencies; we will not attempt to do so in this paper.

5. Formal syntax and semantics

In this section we give the abstract syntax of constraint diagrams, that is, a formal definition which is independent of any topological and visual representations, and a partial formal semantics. For space reasons, the definition of projections is omitted from this paper.

A *constraint diagram* is a tuple

$$\langle \mathcal{C}, \mathcal{C}^d, \beta, \mathcal{Z}, \mathcal{Z}^*, \mathcal{S}^g, \mathcal{S}^e, \mathcal{S}^u, \mathcal{S}^s, \eta, \tau, \zeta, \mathcal{A}, \sigma, \xi \rangle$$

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*. $\mathcal{C}^d$ is a finite set of derived contours.

(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 let $\mathcal{R}' = \mathcal{R} \cup \emptyset$.

(iii) $\mathcal{S}^g$ is a set of *given spiders*, $\mathcal{S}^e$ is a set of *existential spiders*, $\mathcal{S}^u$ is a set of *Universal spiders*, and $\mathcal{S}^s$ is a set of *Schrödinger spiders*. These sets are all pairwise disjoint and let $\mathcal{S} = \mathcal{S}^g \cup \mathcal{S}^e \cup \mathcal{S}^u \cup \mathcal{S}^s$ be a set of *spiders*.

(iv) The function $\eta : \mathcal{S} \rightarrow \mathcal{R}$ returns the habitat of a spider. The function $\tau : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{R}'$ returns the nest of any two spiders, while $\zeta : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{R}'$ is a function that returns the web of any two spiders.

(v) $\mathcal{A}$ is a set of *arrows*. The function $\sigma : \mathcal{A} \rightarrow \mathcal{Z} \cup \mathcal{R} \cup \mathcal{C} \cup \mathcal{S}$ returns the source of an arrow and $\xi : \mathcal{A} \rightarrow \mathcal{Z} \cup \mathcal{C} \cup (\mathcal{S} - \mathcal{S}^u)$ returns the target of an arrow.

We use the value $\perp$ to denote undefined values.

A *model* for a constraint diagram d is a tuple

$$m = \langle \mathbf{U}, \psi, \Psi, \phi \rangle$$

such that $\mathbf{U}$ is a set and ψ, Ψ, ϕ are functions:

- $\psi : \mathcal{S} \rightarrow \mathbf{U} \cup \{\perp\}$ maps spiders to elements of $\mathbf{U}$ or to the special symbol $\perp$, and
- $\Psi : \mathcal{C} \rightarrow 2^{\mathbf{U}}$ maps contours to subsets of $\mathbf{U}$.
- $\phi : \mathcal{A} \rightarrow 2^{\mathbf{U} \times \mathbf{U}}$ maps arrows to relations on $\mathbf{U}$.

We can extend the definition of Ψ to include the interpretation of other elements of d which denote sets in the model:

(i) *Boundary.* The boundary denotes the universal set.
$\Psi(\beta) = \mathbf{U}$.

(ii) *Zones.* The semantics of a zone $z \in \mathcal{Z}$ is determined by which contours enclose it and which don't

$$\Psi(z) = \bigcap_{c \in z \vee c = \beta} \Psi(c) \setminus \bigcup_{c \in \mathcal{C} \setminus z} \Psi(c).$$

By letting the set intersection operation range over the boundary contour, we make sure that even the zone that is external to all contours has a well-defined semantics.

(iii) *Regions.* The value of Ψ of a region is the union of the semantics of the zones in the collection: For $r \in \mathcal{R}$,

$$\Psi(r) = \bigcup_{z \in r} \Psi(z).$$

The following conditions must hold for any compliant model of a constraint diagram.

The **Plane Tiling Condition** ensures that all elements fall within sets denoted by zones:

$$\bigcup_{z \in \mathcal{Z}} \Psi(z) = \mathbf{U}$$

We can derive from this condition that an intersection of contours that doesn't appear as a zone must be empty. Let c_1 and c_2 be two distinct contours in a concrete diagram. Then no zone will contain both c_1 and c_2. So, it follows from the plane tiling condition that any "zone" containing both c_1 and c_2 denotes the empty set. Hence $\Psi(c_1) \cap \Psi(c_2) = \emptyset$. Similarly, if c_1 is contained in c_2 then it follows from that condition that $\Psi(c_1) \subseteq \Psi(c_2)$.

The **Spider Condition** ensures that an element denoted by a spider is in the set denoted by the habitat of the spider:

$$\forall s \in \mathcal{S}^g \bullet \psi(s) \in \Psi(\eta(s))$$

$$\forall s \in \mathcal{S}^s \bullet \psi(s) \in \Psi(\eta(s)) \cup \{\perp\}$$

$$\forall s \in \mathcal{S}^e \bullet \exists x \in \Psi(\eta(s))$$

The **Strangers Condition** ensures that if elements denoted by two distinct spiders are equal then they must fall within the set denoted by their web:

$$\forall s, t \in \mathcal{S}, s \neq t \bullet \psi(s) = \psi(t) \Rightarrow \psi(s) \in \Psi\left(\zeta(s,t)\right).$$

The **Mating Condition** ensures that if the elements denoted by two distinct spiders fall within the set denoted by their nest, then these elements must be equal:

$$\forall s, t \in \mathcal{S} \forall z \in \tau(s,t) \bullet \psi(s), \psi(t) \in \Psi(z) \Rightarrow \\ \psi(s) = \psi(t).$$

The **Shading Condition** maintains that the set denoted by a shaded zone contains no elements other than those denoted by spiders

$$\forall z \in \mathcal{Z}^* \bullet \Psi(z) \subseteq \bigcup_{s \in \mathcal{S}} \{\psi(s)\}$$

Here we adopt the standard convention that a union over an empty range results in the empty set. Together with the spider condition, this condition ensures that the only elements in a set denoted by a shaded zone are the elements represented by any spiders impinging on that zone. Specifically, the set denoted by a shaded zone not containing feet of any spiders is empty.

The **Arrow Condition** ensures that the element or set denoted by the target of an arrow is the image of the element or set denoted by the source of the arrow under the relation denoted by the arrow. It also ensures that navigation expressions are properly quantified.

For each arrow a we define

$$N(a) = (\Psi(\xi(a)) = \Psi(\sigma(a).\phi(a)))$$

$N(a)$ is the "navigation expression" of a. If $\xi(a) \in \mathcal{S}$, then $\Psi(\xi(a)) = \{\psi(\xi(a))\}$, etc. The semantics of a non-quantified arrow a is just $N(a)$. For each arrow a we also define

$$Q(a) = \mathit{if}\ \sigma(a) \in \mathcal{S}^e\ \mathit{then}\ \exists x_{\sigma(a)} \in \eta(\sigma(a))\ \mathit{else} \\ \mathit{if}\ \sigma(a) \in \mathcal{S}^u\ \mathit{then}\ \forall x_{\sigma(a)} \in \eta(\sigma(a))\ \mathit{else}\ []$$

where $[]$ is the empty string. $Q(a)$ is the "quantification expression" for a. We introduced and informally discussed arrow sequences in §4. We define an *arrow sequence* to be a sequence $a_1, \ldots, a_n$ of distinct arrows for which:

$$\begin{gathered}\sigma(a_1) \in \mathcal{S}^e \cup \mathcal{S}^u \wedge (\forall a \in \mathcal{S}^e \cup \mathcal{S}^u \bullet \\ \eta(\sigma(a_1)) \neq \xi(a)) \wedge \forall a_i \in a_2..a_n \bullet \\ (\sigma(a_i) \in \mathcal{S}^e \cup \mathcal{S}^u \cup \mathcal{C}^d \wedge (\sigma(a_i) \in \mathcal{S}^e \cup \mathcal{S}^u \\ \Rightarrow \eta(\sigma(a_i)) \in \xi(a_{i-1})) \wedge \xi(a_{i-1}) \in \mathcal{C}^d)\end{gathered}$$

We define, for $i = 1..n-1$,

$$P(a_i) = Q(a_i) \bullet (N(a_i) \wedge P(a_{i+1}))$$

$$P(a_n) = Q(a_n) \bullet N(a_n)$$

$P(a)$ is a recursive function giving the "partial predicate" of a, that is, the interpretation of a and any arrow dependent on a; it is not necessarily the full predicate for a because it does not include the quantification expression of any arrows earlier in the sequence on which a depends. Thus the semantics of an arrow sequence, $A = a_1, \ldots, a_n$, is $P(a_1)$.

If two or more arrows are sourced on the same quantified spider, then the sequences associated with each arrow are within the scope of the quantifier of that spider. We ensure that the problem cases considered in §4 do not occur by insisting that any arrow is part of an arrow sequence or is not quantified and, if there is more than one sequence in a diagram, then there should be no mixture of types of quantifier, they should all be universal or all existential. In §6 we sketch an idea that should solve most of these problems.

The semantics of a derived contour is just the semantics of the target of any arrow targeted on it. We extend Ψ to derived contours. For all $c \in \mathcal{C}^d$, for all arrows a such that c is the target of a, $\Psi(c) = \Psi(\xi(a))$. With this explicit definition, the sets denoted by derived contours will be properly defined (i.e., with full quantification of the navigation expressions used to define them) when they are used in conjunction with the other semantic conditions.

6. Further work

We have given a partial formal semantics of constraint diagrams and discussed the problems involved in giving a full semantics. Experience of using constraint diagrams suggests that there are some constraints that are expressed much more concisely and intuitively using them; however, it has also highlighted properties that are, at best, awkward to express, without further textual annotation. This has led to ideas on how to use visual constraint notations (eg, constraint diagrams) in combination with textual (eg, OCL) or symbolic (eg, raw predicate logic) and each other using *constraint trees*, informally introduced in [11]. The nodes of constraint trees may be logical expressions, in any notation, or logical connectives. Constraint trees also provide a way of modularizing constraint diagrams, and composing constraints from modules; this allows the notation to be scalable. Constraint trees would allow the ordering of clauses in a constraint; ordering was the problem in most of the issues we discussed in §4. One of the main reasons for developing a formalization of constraint diagrams is to develop diagrammatic reasoning rules for the notation and hence provide a step towards constructing a reasoning system which combines both diagrammatic and textual rules, and which handles issues of modularity.

Acknowledgements We would like to thank Jean Flower and John Taylor for comments on earlier drafts of this paper and also the anonymous referees for their very helpful comments. We acknowledge Yan Sorkin, the creator of the *CDEditor* [4], the automatic tool that was used to generate all the diagrams presented here. Authors Howse and Kent acknowledge some support from the UK EPSRC grant GR/M02606. Part of this research was undertaken while author Gil was at the IBM T. J. Watson Research Center.

References

[1] L. Euler. *Lettres a Une Princesse d'Allemagne*, volume 2. 1761. Letters No. 102–108.
[2] J. Gil, J. Howse, S. Kent. Formalising Spider Diagrams, *Proc. VL99*, Tokyo, Sept 1999. IEEE Press, 130-137.
[3] J. Gil, J. Howse, E. Tulchinsky. Positive semantics of projections. Accepted for JVLC. To appear, 2001.
[4] J. Gil, Y. Sorkin. The Constraint Diagrams Editor, http://www.cs.technion.ac.il/Labs/ssdl/research/cdeditor/.
[5] E. Hammer. *Logic and Visual Information*. CSLI Publications, 1995.
[6] D. Harel. On visual formalisms. In J. Glasgow, N. H. Narayan, B. Chandrasekaran, eds, *Diagrammatic Reasoning*, 235-271. MIT Press, 1998.
[7] J. Howse, F. Molina, J. Taylor. SD2: A sound and complete diagrammatic reasoning system. *Proc. VL 2000*, Seattle, Sept 2000. IEEE Press, 127-136.
[8] J. Howse, F. Molina, J. Taylor. On the completeness and expressiveness of spider diagram systems. *Proc. Diagrams 2000*, Edinburgh, Sept 2000. LNAI 1889, Springer, 26-41.
[9] J. Howse, F. Molina, J. Taylor, S. Kent, J. Gil. Spider Diagrams: A Diagrammatic Reasoning System. Accepted for JVLC. To appear, 2001.
[10] S. Kent. Constraint diagrams: Visualising invariants in object oriented models. In *In Proc. OOPSLA97*, ACM SIGPLAN Notices 32, 1997.
[11] S. Kent and J. Howse. Constraint Trees, in Clark A, Warmer J (eds), *Advances in Object Modelling with OCL*, Springer Verlag. To appear.
[12] R. Lull. *Ars Magma*. Lyons, 1517.
[13] Object Management Group. Unified Modeling Language Specification, Version 1.3. Available from www.omg.org.
[14] C. Peirce. *Collected Papers*. Harvard Univ. Press, 1933.
[15] S.-J. Shin. *The Logical Status of Diagrams*. CUP, 1994.
[16] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag.*, 1880. 123.
[17] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.