Constraint Diagrams: A Step Beyond UML

Joseph (Yossi) Gil
Software Systems Laboratory
Faculty of Computer Science
Technion , Technion City, Haifa
32000, ISRAEL.
yogi@cs.technion.ac.il

John Howse
School of Computing &
Mathematical Sciences
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

The Unified Modeling Language (UML) is a set of notations for modelling object-oriented systems. It has become the de facto standard. Most of its notations are diagrammatic. An exception to this is the Object Constraint Language (OCL) which is essentially a textual, stylised form of first order predicate logic. We describe a notation, constraint diagrams, which were introduced as a visual technique intended to be used in conjunction with the UML for object-oriented modelling. Constraint diagrams provide a diagrammatic notation for expressing constraints (e.g., invariants) that could only be expressed in UML using OCL.

Keywords

Modelling, visual formalism, object-oriented software development, formal methods.

1. Introduction

The uptake in industry of notations for designing systems visually has been accelerated with the recent standardisation of UML. But, in our opinion, UML is only the culmination of the first stage in the development of this young field. It brings together a number of informal visual notations (the possible exception being statecharts) that have proven useful to some parts of industry. It has made little progress in integrating these notations; and it certainly does not include anything that is radically different from the existing status quo (but then the goal was to consolidate not to innovate).

In this paper, we provide some insight into what lies beyond UML. We describe a notation, *constraint diagrams*, which were introduced in (Kent, 1997) as a visual technique intended to be used in conjunction with the Unified Modelling Language for object-oriented modelling. Constraint diagrams provide a diagrammatic notation for expressing constraints (invariants) that could only be expressed using the Object Constraint Language (Warmer and Kleppe, 1998), essentially a textual, stylised form of first order predicate logic which is part of the UML standard (OMG, 1997).

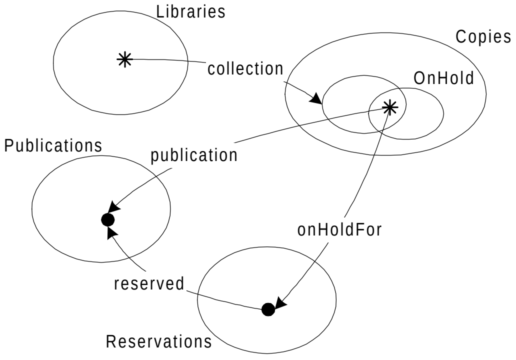


Figure 1 - A constraint diagram

Constraint diagrams are a significant advance on class diagrams in UML for the visualisation of object structures. Whereas class diagrams are only able to show that there are relationships between certain kinds of object, constraint diagrams are able to visualise properties of those relationships and compositions of those relationships. Whereas class diagrams make no use of the relative positions of the diagrammatic elements (e.g. whether a class overlaps with another class or not), the relative positioning of diagrammatic elements on constraint diagrams is vital.

The constraint diagram in Figure 1 expresses (amongst other constraints) an invariant on a model of a library system: for any library object, and any copy of that library which is on hold, that copy's publication must be the same as that associated with the reservation for which it is on hold:

$$\forall x \in \mathit{Libraries}, \forall y \in x.\mathit{collection} \cap \mathit{OnHold},$$
$$\mathit{onHoldFor}.\mathit{reserved} = y.\mathit{publication}$$

This reading is obtained by treating the $*$s as wildcards, universal quantifiers over the regions which contain them, and arrows as showing the range of relations when their domain is restricted to the set or element at their source. Venn diagrams are then used to show relationships between all the sets and elements involved.

Pairs of constraint diagrams have also been used in post and contract boxes to express post conditions and pre/post contracts for actions in a visual form (Kent and Gil, 1998), which forms the basis of further work in three dimensional notations for software and systems modeling (Gil and Kent, 1998). A form of the notation has also been used in the precise, visual representation of design patterns (Lauder and Kent, 1998).

A second goal of this paper is to illustrate how hard it is to define a visual notation, and to highlight some of the issues that arise. 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. And whenever a new piece of notation was added, the impact of that notation on what was already there was not obvious.

A sub-notation of the language of constraint diagrams is the language of spider diagrams – essentially constraint diagrams without the arrows. Spider diagrams are themselves a development from Venn diagrams and Euler circles. The paper is structured in a similar manner. Section 2 overviews the work on Venn diagrams and Euler circles, and places spider diagrams in that context. Section 3 introduces and discusses the informal semantics of the notation for spider diagrams, being careful to motivate and explore the consequences of decisions made. Similarly, Section 4 introduces the notation for constraint diagrams. Section 5 summarises some of the issues still outstanding before the formal definitions of syntax and semantics can be completed.

2. Venn diagrams and Euler Circles

Circles or closed curves, which we will call contours, have been in use for the representation of classical syllogisms since at least the Middle Ages (Lull, 1517). The Swiss mathematician Leonhard Euler (1707-1783) introduced the notation we now call Euler circles (or Euler diagrams) (Euler, 1761) to illustrate relations between classes. This notation uses the topological properties of enclosure, exclusion and intersection to represent the set-theoretic notions of subset, disjoint sets, and set intersection, respectively. Table 1 illustrates the possible relationships between two contours.

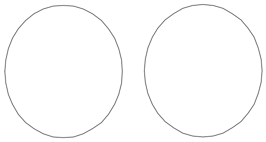
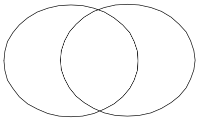
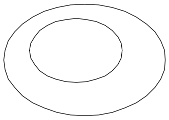
	Contours are disjoint, meaning that the sets they denote are disjoint.
	Contours intersect, meaning that the sets they denote *may* intersect.
	One contour may be contained in another, with the corresponding relationship between the sets they denote.

Table 1: Relationships between contours

The nineteenth century logician John Venn modified this notation to represent logical propositions (Venn, 1880). In Venn diagrams all possible intersections of the closed curves must be shown and shading is used to show that a particular region represents the empty set. Figure 2 shows the standard "clover" Venn diagram of three intersecting contours.

More (1959) gives an algorithm for adding a new closed curve to a Venn diagram. In Figure 3, a new, highlighted, contour has been added to the standard clover. Note that all possible intersections between the four contours occur.

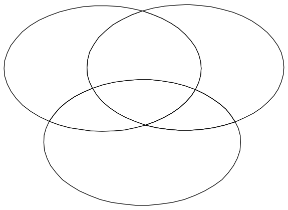


Figure 2 – Clover Diagram

In 1896, the logician Charles Peirce modified Venn diagrams by introducing *X-sequences* to introduce elements and disjunctive information into the system (Peirce, 1933). Very recently formal semantics and inference rules have been developed for Venn-Peirce diagrams (Shin, 1994) and Euler diagrams (Hammer, 1995).

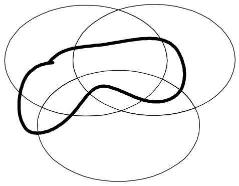


Figure 3 – Four-contour Venn diagram

In summary, Venn-Peirce diagrams are expressive, but complicated to draw because all possible intersections have to be drawn and then some regions shaded. Euler circles are intuitive and easier to draw, but not so expressive because they do not include provisions for shading and for "X-Sequences".

3. Spider Diagrams

Spider Diagrams are Euler circles augmented as follows:

1. *Shaded Regions*, just like in Venn Diagrams.
2. *Spiders*, which are similar to X-Sequences in Venn diagrams, used to denote that an element exists in a set which is the union of one or more regions. They are different from X-sequences because a region might have more than one spider in it, e.g. to denote that a set has two or more elements. Spiders may also be connected by *strands* or *ties* in a region, to indicate that elements denoted by the spiders may or must be the same in that region.
3. *Projections*, which can be used to show the intersection of more than three closed curves in a clear and uncluttered way given the notorious difficulty of showing the intersection of more than three sets on a Venn diagram.

3.1 Contours and Regions

A *contour* is a simple closed plane curve. Contours denote sets of arbitrary size. It is convenient to draw contours as ellipses. However, this is not mandatory. Other iconic representations may be used for making a visual distinction between different kinds of contour. For example, in object-oriented modelling, rectangle contours are used to indicate that a set corresponds to a class of objects. All concepts described in this section are independent of the chosen iconic representations. In Venn diagrams, all contours must intersect. We do not require this property in spider diagrams. As with Euler circles, two contours can stand in one of the three relations listed in Table 1. A *boundary contour* is not contained in and does not intersect with any other contour. We assume that a diagram has one and only one boundary contour, which denotes the universal set for that diagram. However, most of the time we do not bother to draw the boundary contour: it is assumed to be the bounding box for the diagram, be it the edges of the drawing surface, the edges of a figure, etc.

A *basic region* is the bounded region of the plane enclosed by a contour. A *region* is defined as follows: any basic region is a region; if r and s are regions, then their union, intersection, or difference, are regions provided these are non-empty. A *minimal region* is a region having no other region contained within it. Thus a minimal region is an area enclosed by one contour or more which is not further divided by other contours, so the contours of a Spider diagram partition the plane into disjoint minimal regions. Figure 4 shows all but one (the minimal region outside the contours shown but inside the boundary contour is not depicted) of the minimal regions of a standard “clover diagram”.

Regions can be generated by taking the union of any combination of the minimal regions. In a Venn diagram with c contours, there are 2^c minimal regions and $2^{2^c}-1$ regions. Thus, in Figure 4, there are $2^8 – 1 = 255$ regions in total, which is the number of ways the 8 minimal regions (including the one not shown) may be combined. The set denoted by a minimal region is easily calculated: it is the intersection of all contours that contain it, minus the union of all contours that do not contain it.

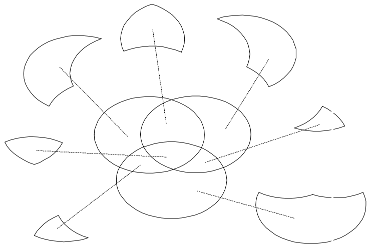


Figure 4 - minimal regions

A *contour label* is a Capitalized string of characters, appearing outside some contour. A *region label* is an underlined, Capitalized string of characters, appearing in a minimal region. Underlining the region label allows region and contour labels to be easily distinguished.

3.2 Spiders

A *spider* is a tree with nodes (called *feet*) placed in different minimal regions; the connecting edges (called *legs*) are straight lines. A spider *touches* a minimal region if one of its feet appear in that region. A spider may only touch a minimal region once. A spider is said to *inhabit* the region which is the union of the minimal regions it touches; this region is called the *habitat* of the spider.

Spiders are used to denote elements. Two distinct spiders denote distinct elements, unless they are joined by a *tie* or by a *strand* (see below). A *spider label* is a lowercase string of characters.

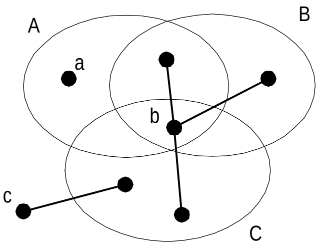


Figure 5 - Spiders

In Figure 5, the spider labelled b inhabits a region which is the union of four minimal regions. The semantics is that

$$b \in (B - C) \cup (C - A)$$
$$a \in (A - B) - C$$
$$c \in (U - A) - B$$

where U is the universal set denoted by the boundary contour. Also, all the elements are distinct, i.e., $a \neq b$, $b \neq c$, and $a \neq c$.

A *strand* is a wavy line connecting two nodes, from different spiders, placed in the same minimal region. Two spiders joined by a strand are referred to as *friends*. The *web* of spiders s and t is the union of minimal regions each containing a strand between nodes of s and t.

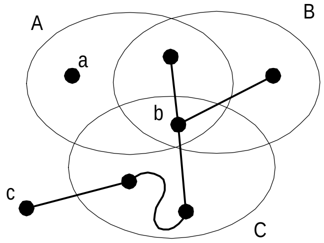


Figure 6 - strands

Strands (of a web) are used to allow spiders to denote the same elements in some circumstances. Specifically, spiders s and t may (not must) denote the same element if that element is in the set denoted by the web of s and t. In Figure 6, it is possible that if the element denoted by c happens to be in C then this may be the same as the element denoted by b. More generally, the elements denoted by s and t are distinct if they are in different minimal regions or not in the web of s and t.

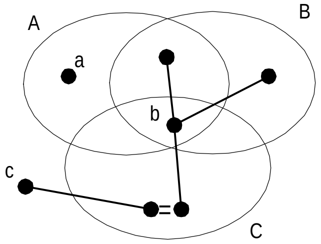


Figure 7 - ties

A *tie* is a double, straight line (an equals sign) connecting two feet, from different spiders, placed in the same minimal region. Two spiders joined by a tie are referred to as *mates*. The *nest* of spiders s and t is the union of minimal regions each containing a tie between feet of s and t. If both the elements denoted by spiders s and t are in the set denoted by the same minimal region in the nest of s and t, then s and t denote the same element. 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.

A tie is stronger than a strand in the sense that it requires s and t to denote the same element in any minimal region in which the tie appears. Thus in Figure 7, $c = b$ when both $c \in (C - A) - B$ and $b \in (C - A) - B$, otherwise $c \neq b$. Clearly, if there is a tie between feet, then a strand between those feet is redundant. Similarly, multiple strands or ties between the same pairs of feet are redundant.

3.3 Shading and Schrödinger Spiders

Already, spiders can be used to place a lower limit on the number of elements in a set. In Figure 8, the region $A - B$ has at least two elements, and the region $B - A$ has at least one element. Clearly one can place any number of spiders in a region, hence express any lower bound.

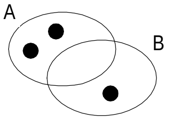


Figure 8 - cardinality with spiders

In order to place an upper bound on the cardinality of a set, we need a way of saying that a region is empty apart from those elements denoted by spiders in the region. We do this by shading. A minimal region is *shaded* if it contains a × or it is actually shaded. A region is *shaded* if each of its component minimal regions is shaded. Shading is visually appealing, but difficult to draw freehand, a × is easier to draw freehand but is, perhaps, not so visually appealing.

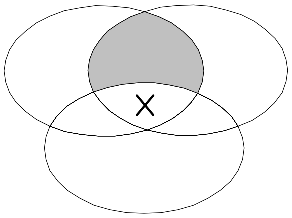


Figure 9 - shaded regions

Figure 9 mixes both mechanisms to show that two minimal regions are empty. It is equivalent in meaning to Figure 10.

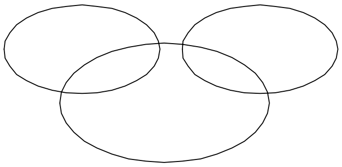


Figure 10 - alternative to shading

Figure 11 shows a case, which is difficult to show without shading. The difficulty is that all three sets within the boundary need to intersect as the intersection of all three may not be empty, but, if one is not careful, this has the effect of introducing new regions. Shading is then required to show that these new regions are empty. So the effect of shading a region and placing no spiders in that region, is to guarantee that the cardinality of the set denoted by the region is zero.

Shading a region which includes spiders has the effect of placing an upper limit on the number of elements in the set denoted by the region. In Figure 12, A contains at most 3 elements; it may contain less as the elements denoted by spiders *a*, *b* and *c* may be selected from other regions. B contains exactly 2 elements; the spiders in the region mean a lowerbound on its size of 2, the shading ensures that this is also an upperbound. C contains between 1 and 3 elements.

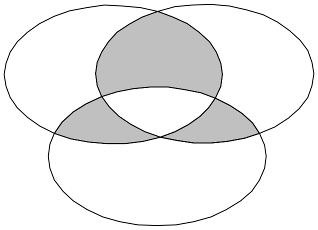


Figure 11 – Shading is essential

In Figure 12, the sizes of sets A and C are related: the more elements in A the less in C, and vice-versa. If we wished to avoid this, then it would be necessary to always have an element in the universal set but not in the sets represented by all other contours. In Figure 13, the same restrictions on the size of A and C are in force, but this time if *a, b* and *c* denote elements in A this has no impact on the size of C, as the habitats of these spiders do not include C. But the price to pay is that D must contain a single element if the size of A and C both hit their lower bounds.

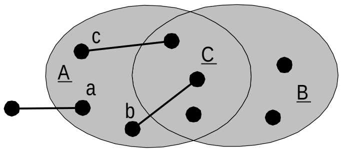


Figure 12 - Shading & spiders

We are not sure whether this has any practical significance when using spider diagrams in modelling (except perhaps that it's also awkward to draw). However, as mathematicians we feel uneasy about such a state of affairs. The fix is not difficult: a *Schrödinger Spider*.

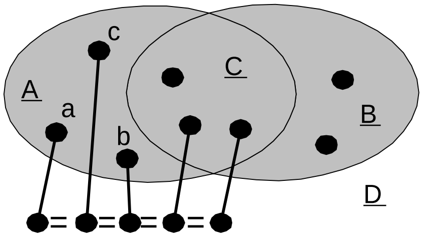


Figure 13 - Shading & spiders II

A *Schrödinger Spider* is represented by the symbol ⊘, and may appear in any region. It can be friends or mates with other spiders. A Schrödinger spider denotes a set whose size is zero or 1: rather like Schrödinger's cat one is not sure whether the element exists or not. Figure 14 is a reworking of Figure 13, this time using Schrödinger spiders. *D* is not forced to contain an element. The diagram is also less complicated to draw.

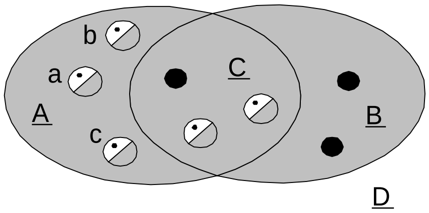


Figure 14 - Schrödinger Spiders

Obviously there is a limit to the amount we can express in this notation about the cardinality of sets. For example, we are unable to say that the size of *A* is the sum of the sizes of *B* and *C*. In the past, we have toyed with extending the notation to allow this kind of constraint to be expressed (Kent and Gil, 1998), but have realised that all we were really doing is adding textual annotations to the diagram. We can use labels to achieve a similar effect: for example, it is easy to write $|A| = |B| + |C|$. Thus, in general, labels allow us to combine constraints expressed visually with constraints expressed textually; diagrams can then be used for those constraints which are intuitive to express using a diagram, and do not need to be overburdened with textual annotations in an attempt to make them more expressive, but which results in them being overly complex with corresponding loss of intuitiveness.

3.4 Projections

A *projection* is a contour, which is dashed in appearance. A *projected label* is a contour or region label, written within brackets and appearing outside some projection.

The *region containing a contour* is the smallest region that *strictly* contains the basic region for that contour. Strictness ensures that the basic region itself is not the region containing the contour. A projection denotes the set obtained by intersecting the set denoted by its projected label with the set denoted by its containing region, which can always be calculated from the sets denoted by contours other than the projection itself.

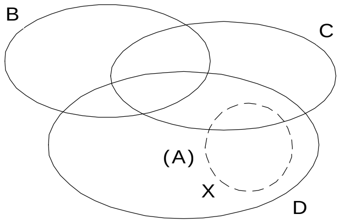


Figure 15 - Simple projection

Figure 15 shows a simple example. The dashed contour labelled *X* denotes the set obtained by "projecting" the set *A* onto the containing region *D* – *B*, i.e., $X = A \cap (D - B)$.

The same semantics could have been obtained by using More's algorithm (More 1959) to draw the Venn diagram with four contours, as in Figure 16, in which $X = \underline{X1} \cup \underline{X2} = A \cap (D - B)$.

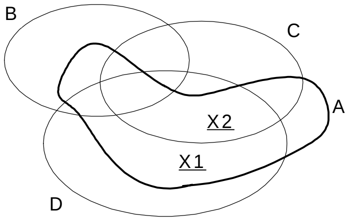


Figure 16 – Semantics of Figure 15

There are fascinating mathematical intricacies involving interacting projections.

4. Constraint Diagrams

Constraint diagrams are spider diagrams augmented by *arrows* and *wildcards*. Arrows determine relationships between sets and wildcards denote universal quantification.

4.1 Arrows

An arrow has a label, a source and a target. The source of an arrow can be a contour, a region or a spider; it is the set or element from which *navigation* (i.e., the relationship computation) begins. In figure 18 the source of arrow *f is* spider *x* and its target is contour *B*. The semantics of the navigation expression is $x.f = B$, where $x.f$ is shorthand for $\{y : f(x, y)\}$.

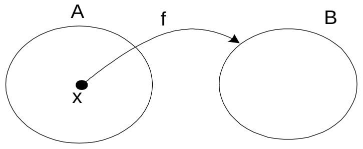


Figure 18 – One-arrow constraint diagram

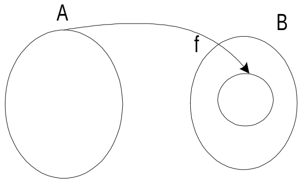


Figure 19 – Derived contour

In Figure 19 the source of arrow *f is* spider *x* and its target is a contour contained in *B*. In this case, we have $A.f \subseteq B$. The dot notation is overloaded; $A.f$ is shorthand for applying *f* to each element in *A* and then taking the union of the resulting sets; i.e., it is the union of sets $\{y : f(x, y)\}$ for each $x \in A$. The contour on which *f* is targeted is a *derived* set; it is defined by the arrow *f*.

The source of an arrow can also be a region. In Figure 20, the arrow has a double source indicating that the source of the arrow is the union of the two minimal regions denoting the sets $A - B$ and $B - A$. The target of an arrow should be a set. However, the arrow *f* in Figure 20 is targeted on a spider. This spider is treated semantically as a derived singleton set. The interpretation of the navigation expression is $((A - B) \cup (B - A)).f \in C$. We are implicitly allowing coercion between a singleton set and its element.

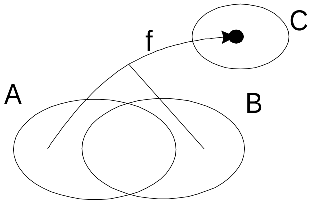


Figure 20 – Region as source and spider as target of an arrow

4.2 Quantification

Wildcards are introduced into constraint diagrams to represent universal quantification. The wildcard spider $*$ ranges over all the elements of the set denoted by the minimal region in which it is situated.

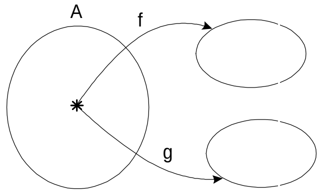


Figure 21 – Universal quantification

In Figure 21, the wildcard 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., $\forall x \in A \bullet x.f \cap x.g = \{\}$.

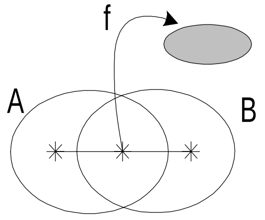


Figure 22 – Articulated wildcard spider

A wildcard can only be the source of an arrow. No arrow can be targeted on a wildcard and a wildcard that is not the source of an arrow cannot exist. A wildcard can be a foot of an articulated spider (i.e., a spider with more than one foot), in which case all feet of the spider are wildcards.

In Figure 22, the source of arrow *f* is an articulated, wildcard spider. The interpretation of this is that for each *x* in $A \cup B$, *x.f* is empty, i.e., $\forall x \in A \cup B \bullet x.f = \{\}$.

The navigation expression in Figure 23 is existentially quantified. Its interpretation is that there exists an *x* in *A* such that *x.f* is not empty, i.e., $\exists x \in A \bullet x.f \neq \{\}$.

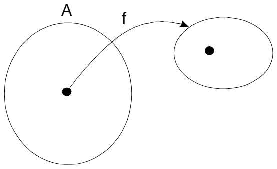


Figure 23 - Existential quantification

4.3 Navigation

So far, we have only considered single-arrow navigation expressions. In Figure 24, there is a two-arrow navigation expression *x.f.g*. This is interpreted as {*y* : *g*(*x.f*, *y*)}; we apply *g* to each element of the derived set *x.f*.

Furthermore, the set obtained by navigating from *x* via arrows *f* and *g* is the same as that obtained by navigating from *x* along *h*. Thus we have *x.f.g* = *x.h*.

In Figure 1, we had a similar expression involving a constraint on a library system. One of the key strengths of constraint diagrams is their ability to illustrate diagrammatically navigation expressions and the relationships between them.

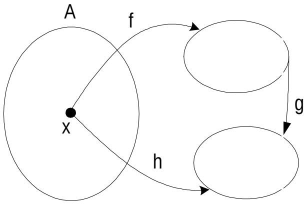


Figure 24 - Navigation

5. Issues and Further Work

The formal semantics of spider diagrams, with the exception of projections, are given in (Gil, Howse, Kent 1999). Diagrammatic inference rules have been developed for spider diagrams along with rules for combining spider diagrams (Howse, Molina, Taylor, Kent 1999).

There are, however, some difficult issues to be considered before the formal semantics of constraint diagrams can be given fully. As mentioned earlier, there are some fascinating mathematical intricacies involving interacting projections. For example, consider Figure 25. *X* and *Y* are the labels of the projected contours and *A* and *B* are the projected sets. How do we interpret *X* and *Y*?

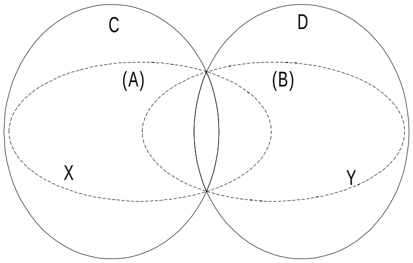


Figure 25 – Interacting projections

By the definition of projected contours, we have two simultaneous set equations in *X* and *Y*:

$$X = A \cap (C \cup Y)$$
$$Y = B \cap (D \cup X)$$

These can be solved using a form of Gaussian elimination and some results from set theory. Unfortunately, there are many possible solutions. There is, however, a unique “minimal solution” in this case:

$$X = A \cap (C \cup B)$$
$$Y = B \cap (D \cup A)$$

This is the intuitive solution. The interpretation of interacting projections in general is still being investigated.

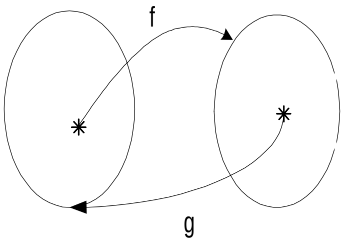


Figure 26 - Circularity

Another difficult issue is that of circularity. Consider Figure 26. Each contour is a derived contour defined in terms of the other contour. It is very difficult to interpret this and it is doubtful whether such a situation should be allowed.

A related difficulty is that of the ordering of quantifiers. In Figure 27, there are at least two logically different interpretations: $\forall x \in A, \exists y \in B \bullet x.f = y.g$ and $\exists y \in B, \forall x \in A \bullet x.f = y.g$. These are very different. In the second interpretation, it is the same *y* for each *x*, while in the first, each *x* may be associated with a different *y*.

We are developing a very sophisticated algorithm for capturing the intuitive semantics of a constraint diagram and translating it into a logical formula. One way of tackling the problems of circularity and the ordering of quantifiers is to insist that each quantifier is at a particular unique "level", and that any diagram not satisfying this condition is deemed to be not well-formed.

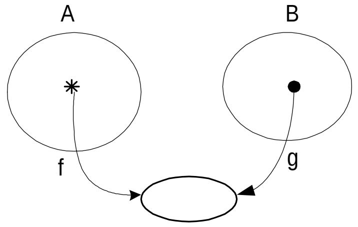


Figure 27 – Ordering of quantifiers

References

Euler, L. (1761) Lettres a Une Princesse d'Allemagne, Vol 2, Letters No. 102-108.

Gil, Y, Howse, J, Kent, S (1999) Formalizing Spider Diagrams, *Proceedings of VL99*.

Gil, Y. and Kent, S. (1998) Three Dimensional Software Modelling. *Proceedings of International Conference in Software Engineering 1998,* IEEE Computer Society Press.

Hammer, E.M. (1995) *Logic and Visual Information,* CSLI Publications.

Harel, D. (1998) On Visual Formalisms. In: Glasgow, J., Hari Narayanan, N. and Chandrasekaran, B., (Eds.) *Diagrammatic Reasoning*, pp. 235-271. MIT Press.

Howse, J, Molina, F, Taylor, J, Kent, S (1999b) Reasoning with Spider Diagrams, *Proceedings of VL99*.

Kent, S. (1997) Constraint Diagrams: Visualising Invariants in Object Oriented Models. *Proceedings of OOPSLA97,* ACM Press.

Kent, S. and Gil, Y. (1998) Visualising Action Contracts in OO Modelling. *IEE Proceedings: Software* 145.

Lauder, A. and Kent, S. (1998) Precise Visual Specification of Design Patterns. *Proceedings of ECOOP98*, Springer Verlag.

Lull, R. (1517) *Ars Magma*, Lyons.

OMG (1997) UML 1.1. Specification. *OMG Documents ad970802-ad970809.*

Peirce, C. (1933) *Collected Papers*, Harvard University Press.

Shin, S.-J. (1994) *The Logical Status of Diagrams*, CUP.

Venn, J. (1880) On the Diagrammatic and Mechanical Representation of Propositions and Reasonings. *Phil.Mag.* 123.

Warmer, J. and Kleppe, A. (1998) The Object Constraint Language: Precise Modeling with UML, Addison-Wesley.