Evaluating and Generalizing Constraint Diagrams

Gem Stapleton *,1,2 , Aidan Delaney 2

Visual Modelling Group, University of Brighton, Brighton, UK
www.brighton.ac.uk/cmis/research/vmg

Abstract

The constraint diagram language was designed to be used in conjunction with the Unified Modelling Language (UML), primarily for placing formal constraints on software models. In particular, constraint diagrams play a similar role to the textual Object Constraint Language (OCL) in that they can be used for specifying system invariants and operation contracts in the context of a UML model. Unlike the OCL, however, constraint diagrams can be used independently of the UML. In this paper, we illustrate a range of intuitive and counter-intuitive features of constraint diagrams and highlight some (potential) expressiveness limitations. The counter-intuitive features are related to how the individual pieces of syntax interact. A generalized version of the constraint diagram language that overcomes the illustrated counter-intuitive features and limitations is proposed. In order to discourage specification readers and writers from overlooking certain semantic information, the generalized notation allows this information to be expressed more explicitly than in the non-generalized case. The design of the generalized notation takes into account five language design principles which are discussed in the paper. We provide a formalization of the syntax and semantics for generalized constraint diagrams. Moreover, we establish a lower bound on the expressiveness of the generalized notation and show that they are at least as expressive as constraint diagrams.

Key words: constraint diagrams, precise software specification, diagrammatic specification, expressiveness

* Corresponding author

Email addresses: g.e.stapleton@brighton.ac.uk (Gem Stapleton), a.j.delaney@brighton.ac.uk (Aidan Delaney).

1 This research is supported by a Leverhulme Trust Early Career Fellowship and by the EPSRC grant EP/E011160/1 for the Visualization with Euler Diagrams project.

2 We would like to thank John Taylor and the anonymous reviewers for their helpful comments on an earlier version of this paper.

Preprint submitted to Elsevier 2 April 2008
1 Introduction

Visual languages play an important role in the design and implementation of software. For example, the Unified Modelling Language (UML) is now a widely adopted industry standard visual notation designed specifically for use by software engineers and is used throughout the software development process, from capturing domain requirements through to implementation. Under some circumstances (such as in a safety critical environment; see, for example, [34]) it is desirable, perhaps even essential, to produce formal models of software. In part, such application areas serve to motivate the need for the precise specification of the UML at both a syntactic and semantic level; the pUML group was set up with this goal in mind [33].

Formal constraint languages bring with them the ability for users to reason in a sound way, thus allowing models to be fully explored, highlighting any unintended behaviour and revealing inconsistencies; constraints have the potential to add significant understanding and rigour to software models. Furthermore, precise models allow implementations to be proved correct. The formal constraint languages in current use, such as the Object Constraint Language (OCL), are symbolic, despite the diagrammatic representations of models to which constraints need to be applied. Despite important benefits, software engineers seem reluctant to use mathematically-based symbolic languages. By contrast, software engineers have readily adopted visual modelling languages. Hence, visual constraint languages which are carefully designed may be more accessible to software engineers; consequently, the provision of such languages may yield an increase in the use of formal software specification.

Part of the creation of a formal software model is likely to involve specifying constraints such as system invariants and operation contracts which, within the UML, is achieved by using the OCL. The OCL is the only purely textual part of the UML and, therefore, does not fit with the UML’s diagrammatic theme. In order to overcome this restriction to using a textual notation for enforcing these types of constraint, Kent introduced constraint diagrams [21] which are designed to complement the visual components of the UML but which can also be used independently of the UML. In figure 1 there is a constraint diagram which expresses an invariant that we might wish to place on a video rental store system: every member can only borrow films that are in the collections of the stores which they have joined. The semantics of constraint diagrams will be explained more fully later, but the asterisk is a universal quantifier, the arrows allow us to make statements about binary relations and the closed curves represent sets (or classes).

At first glance, constraint diagrams appear intuitive and, perhaps, unambiguous, but it was not until a formalization of their semantics was attempted that
a range of ambiguities were noticed [13]. Indeed, only when a formalization was eventually obtained [9] did the complexity of interpreting these diagrams become apparent. We shall demonstrate that, whilst in many examples constraint diagrams are very intuitive (in terms of being, borrowing a phrase from [17], “well-matched to meaning”), there are many situations where their intuitiveness breaks down. Quoting from [17]

Studies such as [14,31,36] have indicated that the most effective representations are those which are well-matched to what they represent, in the context of particular reasoning tasks.

We argue that, in the context of placing formal constraints on models, being well-matched to that which is represented should be considered in terms of both the ease with which the constraint is accurately interpreted by the reader and the ease with which the constraint was enforced by the writer. In the software engineering context, the writer is likely to be a software engineer, but the readership of the constraints is more broad, including other software engineers, domain experts, programmers, and possibly end users (in fact, all those involved in the modelling and implementation process). We will show, by a series of examples, that constraint diagrams are not always well-matched to their semantic interpretation.

There are various situations where reasoning will need to be performed when using formal methods. First, there is reasoning about the model; for example, when one wishes to show that the post-condition of one operation implies the pre-condition of another. Secondly, a programmer will need to determine an appropriate implementation that conforms to the specification which, according to current industry practise, requires some informal reasoning. Thirdly, at a later stage, one might also wish to prove formally that the implementation does indeed conform to the model. Formal reasoning has been investigated for constraint diagrams [8] but we will argue by example that there are instances where intuitive reasoning is unsound which could lead to errors both in specifications and in proofs unless care is taken.
An aim of this paper is to explore the counter-intuitive aspects of the constraint diagram language and to propose a generalized version of these diagrams which overcomes the problems noted. Section 2 provides a brief overview of constraint diagrams. Some intuitive features of constraint diagrams are highlighted in section 3; we consider the individual types of syntactic elements (such as arrows) in terms of their well-matchedness. Problems with the notation arise when the syntactic elements are combined together to form diagrams. We use the phrase *syntax interaction* to refer to the ways in which the syntactic elements work together to make statements. In particular, we can have what we call positive syntax interaction and negative syntax interaction. We show that negative syntax interaction occurs in constraint diagrams by demonstrating some counter-intuitive features in section 4. Our generalized constraint diagram language is presented informally in section 5, where we show that it overcomes the problems noted. The principles we followed when designing the generalized notation are outlined in section 6. Sections 7 and 8 provide a formalization of generalized constraint diagrams and section 9 illustrates how we have followed the design principles. A lower bound on the expressiveness of generalized constraint diagrams is established in section 10 where we also show that we have at least the expressive power of constraint diagrams.

2 An Overview of Constraint Diagrams

In this section we will briefly outline the syntax and semantics of constraint diagrams; a formal treatment can be found in [9]. The so-called *unitary diagram* in figure 1 contains three *given contours*; these are closed curves labelled *Member*, *Film* and *Store* respectively. In a unitary diagram, no two distinct contours have the same label. Given contours represent sets and their spatial relationship is used to make statements about containment and disjointness of sets; in figure 1, the given contours assert that the sets *Member*, *Film* and *Store* are pairwise disjoint because the contours do not overlap in any way. Also in the diagram are three *derived contours*; these are the closed curves that do not have labels and happen to be targeted by *arrows*.

The asterisk is called a *universal spider* and its *habitat* is the *region* in which it is placed. In other words, its habitat is the region inside the contour labelled *Member*. In this example, this region is also called the *domain* of the universal spider; the domain is the region which represents the set that the spider quantifies over, in this case the set *Member*. It is not necessarily the case that the habitat equals the domain and this will become clearer below. In general,

---

3 Constraint diagrams are based on Euler diagrams. The set of given contours in a constraint diagram is called its *underlying Euler diagram*.
A region is a set of zones. A zone is a maximal set of points in the plane that can be described as being inside certain contours (possibly no contours) and outside the rest of the contours. The diagram in figure 1 contains seven zones; for example, there are three zones inside the contour labelled Film.

When developing a language it is beneficial to incorporate syntax familiar to users, provided its use is consistent. Arrows are frequently used to represent relationships in modelling notations, such as class diagrams. The use of asterisks to represent universal quantification is motivated by the use of asterisks in the UML. For example, sequence diagrams use “the traditional way of modeling a loop using an iteration marker, an asterisk” [1]; we can think of universal quantification as somehow iterating through the elements, x, of the set over which we are quantifying because, in the semantics, we consider all possible values of x when determining satisfaction.

In a unitary diagram, every arrow is sourced on a spider or a contour (given or derived) and targets a spider or a contour. Moreover, every arrow has a label and, unlike given contours, two (or more) arrows can have the same label. Derived contours are always the target of at least one arrow and represent sets; the actual set represented is determined by the targeting arrow(s). In figure 1, given any member, m, the derived contour inside the given contour labelled Store represents the image of the relation joined when the domain is restricted to m (i.e. the derived contour represents the set of stores that m has joined).

Unitary diagrams can also contain existential spiders and shading and, strictly speaking, every unitary diagram is augmented with a reading tree (the diagram in figure 1 does not have a reading tree). Existential spiders are denoted by trees whose nodes are round and filled (as opposed to asterisks in the universal case) and they represent the existence of elements. For example, in figure 2, there are two existential spiders; these are the dots labelled y and z and each of these two spiders has a single zone habitat. By contrast, the universal

![Diagram](image-url)
spider labelled x has a two zone habitat and quantifies over the set English ∪ OtherLang. Shading allows us to place upper bounds on set cardinality: in a shaded region, all of the elements are represented by spiders. So, in figure 2, the diagram expresses that the set Film − (English ∪ OtherLang) is empty since there are no spiders placed in the shaded zone. The meaning of the diagram is affected by the order in which the quantifiers (i.e. the spiders) are interpreted. The reading tree above the diagram tells us the order in which to read the quantifiers, thus resolving ambiguities: we start with the universal spider and then we can read the existential spiders in either order and independently of each other. Moreover, the reading tree also provides quantifier scoping and bracketing information (the details of which can be found in [9]). The diagram expresses:

(1) Film, Title and Actor are pairwise disjoint,
(2) English and OtherLang form a partition of Film and
(3) every film, x, has a unique name which is a title and, in addition, x has at least one lead actor.

The first two statements capture the information provided by the underlying Euler diagram. More formally, the diagram is interpreted as:

∀x ∈ English ∪ OtherLang(∃y {y} = {x}.name ⊆ Title ∧
∃z {z} ⊆ {x}.leadActor ⊆ Actor) ∧ PTC

where PTC (which stands for the Plane Tiling Condition) captures the information provided by the underlying Euler diagram and {x}.name is the image of name when the domain is restricted to {x} (similarly for {x}.leadActor). When interpreting diagrams in examples, we will not explicitly state the plane tiling condition even though it forms part of the diagram’s meaning.

The augmented diagram (i.e. a unitary diagram together with a reading tree) in figure 3 expresses, in addition to the information provided by the underlying Euler diagram, that there is a film in every store’s collection and every store stocks at least two copies of that film. More precisely, the diagram expresses:

\[
\forall x \in \text{English} \cup \text{OtherLang} (\exists y \{y\} = \{x\}.name \subseteq \text{Title} \land \\
\exists z \{z\} \subseteq \{x\}.\text{leadActor} \subseteq \text{Actor}) \land \text{PTC}
\]
∃p ∈ Film\{p\}.allCopies ⊆ Stock \land \left(\forall q ∈ Store\{p\} ⊆ \langle q \rangle.collection ⊆ Film \land \langle q \rangle.storeCopies ⊆ Stock \land \exists r, s\{r, s\} ⊆ \langle q \rangle.storeCopies \cap \{p\}.allCopies \land r \neq s)\right) \land PTC.

In general, unitary diagrams can be augmented with different reading trees, some of which give rise to semantically different interpretations. As mentioned above, the reading tree indicates in which order to read the spiders, providing a partial order on them. Not every partial ordering of the spiders gives rise to a valid reading tree, however. The unitary diagram in figure 4 can be augmented with either of the two valid reading trees to the left of the diagram but not with either of the two invalid reading trees on the right. The two valid trees give rise to semantically different meanings. The topmost invalid tree is not valid because q and r are not in a common path and, thus, they are treated as independent when interpreting the diagram but we need to assert their distinctness (i.e. the semantics must state q \neq r). In the other invalid tree, p and s are not in a common path and, as above, are treated as independent but we need to assert that s \not\in \{p\}.allCopies.

To conclude this section, we note that constraint diagrams can be joined together using logical connectives \land, \lor and \lnot to form compound diagrams. That is, if d_1 and d_2 are constraint diagrams then (d_1 \land d_2), (d_1 \lor d_2) and \lnot d_1 are also constraint diagrams.

3 Intuitive Features: Individual Pieces of Syntax

An aim of this paper is to develop a generalized version of the constraint diagram language which overcomes some of their counter-intuitive features. When designing such a generalized language, it would be useful if we can retain the intuitive aspects of constraint diagrams. In this section, we outline some features of constraint diagrams that are well-matched to their meaning. Constraint diagrams use Euler diagrams as a basis and augment them with
various pieces of syntax, namely arrows, spiders, shading and reading trees. We consider each of these pieces of syntax in turn with regard to well-matchedness.

A well-matched language will allow semantic relationships to be mirrored at the syntactic level. For example, the subset relation is mirrored, in Euler diagrams, by the containment of one closed curve by another. Furthermore, the transitive property of the subset relation is also mirrored by containment of closed curves. This leads to free rides \[24,25\] (sometimes called cheap rides \[15\]) which helps to explain some of the advantage diagrams have over symbolic notations. A free ride is essentially an inference that is explicit within a diagram, but would have to be derived when using symbolic notations. For example, the diagram in figure 5 expresses that English $\subseteq$ Film by placing English inside Film. Likewise, StarTrekMovies $\subseteq$ English. The placement of StarTrekMovies inside English automatically places it inside Film thus giving StarTrekMovies $\subseteq$ Film for ‘free’.

![Diagram](Fig. 5. Free rides.)

It is commonly accepted that Euler diagrams are good at making statements about the relationship between sets by exploiting topological properties of enclosure, exclusion and intersection. The use of Euler diagrams in a large variety of areas justifies our claim of their effectiveness; examples of their use include visualizing set-based statistical data \[5,22,23\], displaying the results of database queries \[35\], representing non-hierarchical computer file systems \[6,7\] and viewing clusters which contain concepts from multiple ontologies \[18\]. Thus, a feature of constraint diagrams that is well-matched to meaning is their being based on Euler diagrams.

The property of $x$ being related to $y$ is a directed relationship which can be mirrored by the use of an arrow from the syntactic object representing $x$ to that representing $y$. The use of a chain of arrows (linked by a shared source and target) allows complex navigation expressions to be relatively easily constructed. We believe that arrows are a natural way of expressing (directed) relationships between objects. Thus, we argue that arrows are well-matched to their meaning (i.e. providing information that the source is related to the target).

The placement of objects inside regions to represent elements of the sets represented by those regions also seems well-matched to meaning. Existential
spiders are trees, placed in regions of the diagram and represent elements. The set represented by the region in which the spider is placed contains the element represented by the spider. Here, containment at the semantic level is mirrored by containment at the syntactic level, illustrating well-matchedness. However, we will demonstrate later that this well-matchedness does not always occur due to subtle semantic issues. The distinctness of two elements at the semantic level is mirrored by representing them with distinct syntactic objects (i.e spiders). The diagram in figure 6 asserts, in addition to the information provided by the underlying Euler diagram, that there exist two elements, one of which is in the set $\textit{onShelf}$ and the other is in $\textit{onLoan} \cup \textit{onShelf}$ (we omit the reading tree since the order in which these spiders are read does not impact the semantics). We remark that an existential spider whose habitat is

more than one zone provides disjunctive information. In figure 7, the existential spider inhabits two zones and represents an element either in the set $\textit{OnLoan} \cap \{x\}.\textit{allCopies}$ or in $\textit{OnShelf} \cap \{x\}.\textit{allCopies}$.

Arguments similar to those used for existential spiders show universal spiders are also reasonably well-matched to meaning in that they are placed in the regions that represent the sets of which they are elements. We will show that, as with existential spiders, well-matchedness need not always arise when considering the set represented by the habitat versus the set over which we are quantifying. We will further demonstrate that there are subtle differences with the ways in which we can reason with the two types of spider.

Shading, whilst probably not being as intuitive as other aspects of the constraint diagram language, is still reasonably well-matched to its meaning.
Shading ‘fills out’ the region in which it is placed, indicating that nothing else is present.

Using shading enables more effective diagrams to be drawn by allowing topological and geometric properties to be exploited, such as using simple closed curves, no concurrency and only transverse crossings [29]. These properties are desirable since they aid in reducing the potential for user misinterpretation which can result in unsound inferences. To illustrate, in figure 8 the shaded region can be removed from $d_1$ to give $d_2$. Clearly, diagram $d_1$ is more appealing visually than $d_2$ where it is hard, if not impossible, to identify the individual closed curves from the drawing alone. Thus, the meaning of $d_2$ is not readily available to the readers of the diagram.

Finally, reading trees are used to indicate quantifier reading order and scope. The directed edges in the trees are a natural way of providing the quantifier reading order. The scope of a quantifier is indicated by nodes that are in the paths from the node associated with that quantifier. Reading trees exhibit this information well, with quantifiers in different paths being interpreted independently. The actual formula that is within the scope, however, needs to be derived from the diagram as well; given any formula, it is not necessarily visually apparent whether the formula is within the scope of a quantifier.

In summary, all of the individual pieces of unitary diagram syntax are, at the very least, reasonably well-matched to their intended meaning. As we will demonstrate, however, it is the (sometimes subtle) ways in which the pieces of syntax interact that can give rise to counter-intuitive features. One of our aims is to overcome the counter-intuitive features by providing a generalized version of constraint diagrams and, where possible, we wish to retain the well-matched features. Consequently, we design the generalized notation taking into account the discussions in this section.
4 Counter-Intuitive Features and Negative Syntax Interactions

As mentioned in the introduction, when investigating how well matched constraint diagrams are to meaning, we should consider the ease with which constraints can be read, can be written and, finally, whether intuitive reasoning matches sound reasoning. We consider each of these three facets in turn, highlighting various counter-intuitive aspects of constraint diagrams by way of a series of examples. We summarize this by saying that the constraint diagram notation has some negative syntax interactions.

4.1 Reading

The diagram in figure 9 asserts that, for each member, the films which can be borrowed are available to that member. In this example, the domain of the universal spider labelled $x$ is the whole of Member, but its habitat is a single zone inside one of the derived contours. By contrast, the other universal spider does quantify over its habitat. If we were to redraw the diagram so that the habitat of $x$ was indeed the two zones inside Member then the meaning would be changed: such a diagram would no longer assert that each film which can be borrowed is available to the member. This difference between domain and habitat leads to the diagram not being particularly well-matched to meaning, especially given the symmetry present. This problem also occurs in figure 3, where $p$ quantifies over the set Film, but its habitat does not necessarily represent this set. This counter-intuitive feature is not dependent on spider type. Here, we have observed that there is negative interaction between spiders and derived contours. This type of negative syntax interaction (domain differing from habitat) is unavoidable in constraint diagrams, since quantifiers, $Q$, are always placed in a unitary diagram which, in turn, must contain all of the syntax within the scope of $Q$.

In figure 10, in the left-hand diagram, the set Actor represents the empty set because of the shading. Studying the other diagram might, then, lead one to conclude that the set Actor contains exactly one element due to the existential spider and the shading. However, the reading tree informs us that
the existential spider is read after the universal spider. Consequently, we only know that the set Actor contains exactly one element when the set Film is not empty. Informally, the shading in the first diagram is read independently of the universal spider but not in the second diagram (again, we refer the reader to [9] for a precise formulation of the semantics). Here, we have a negative syntax interaction between spiders, the reading tree and shading.

Each of the two examples given so far in this subsection illustrate a lack of modularity to constraint diagram interpretation. We have to decipher spiders’ domains and the order in which to read the diagram syntax from the reading tree, including syntax which is not explicitly in the tree such as shading. This reading order can have profound impact on what we can deduce from a diagram. The reading tree could be modified to indicate explicitly when to read each piece of syntax but this would result in unwieldy trees, probably resulting in an unusable notation.

To conclude this subsection, we briefly consider the example in figure 3. There is a preference for native English speakers to start reading at the left most spider, to prioritize reading spiders whose domain equals their habitat and to ‘follow arrows’ [10]. The reading tree for figure 3 forces the user to violate all of these preferences: $p$ is not the left most spider; unlike $q$, $p$ has habitat not equal to domain; ‘following the arrows’ would also tell us to read $q$ before $p$. The empirical study in [10] presents subjects with unaugmented constraint diagrams with the hope of ascertaining which readings are intuitive. The added use of a reading tree is likely to reinforce the ‘follow the arrows’ preference, since the reading tree uses arrows to indicate reading order.

4.2 Writing

There are various expressiveness restrictions with the constraint diagram language: they form a first order language which precludes the formalization of those invariants and operation contracts that necessarily involve higher-order aspects; additionally, some relatively simple first order statements are not easily expressed. As a system invariant, we might wish to assert that every film has a lead actor or no actors. Taking the disjunction of the two diagrams in figure 11 may represent an attempt at enforcing this invariant, but does
not achieve what is required: the disjunction expresses that all films have a
lead actor or all films have no actors which is not equivalent to the required
invariant. It is not clear at all whether any constraint diagram can enforce
the required invariant (up to semantic equivalence). The issue here is that we
wish to make a universally quantified, disjunctive statement involving binary
relations. Unitary constraint diagrams do not contain facilities for making
disjunctive statements involving binary relations but quantification can only
occur within unitary diagrams. Even if there is a constraint diagram that en-
forces this invariant up to semantic equivalence the example still illustrates
that any such diagram will not be well-matched to the meaning intended by
the invariant writer.

Perhaps more fundamentally, constraint diagrams have problems making state-
ments that involve equality between elements. This is because, within a unitary
diagram, all spiders are taken to represent distinct elements and quantification
can only occur within such diagrams. To illustrate, we might want to assert
that every film has a lead actor and a director. The actor and director could
be the same person or different people. A failed attempt to express this con-
straint can be seen in figure 12, where the diagram expresses that every film
has an actor and a director who are necessarily different people (since they
are represented by distinct spiders); the actor and director are prevented from
being the same person. Hence, we have spiders negatively interacting from the
point of view of expressiveness, but positively interacting from a well-matched
perspective.

Constraint diagrams without any universal spiders, arrows and derived con-
tours are called spider diagrams [19]. We note that in some spider diagram
systems, pieces of syntax called \textit{strands} (represented by wiggly lines) and \textit{ties}
(represented by an equals sign) are used to assert that two represented ele-

Fig. 11. Disjunction issues.

Fig. 12. Equality issues.
ments might be equal or are equal, respectively [12]. However, these syntactical devices were not incorporated into the formalization of constraint diagrams given in [9]. There are perhaps various reasons for this. We speculate that it may have been to simplify the presentation of the semantics for augmented constraint diagrams. Moreover, there are many intuitive interpretations of stands and ties which are not consistent with each other, particularly in cases where connected spiders inhabit more than one zone; we omit the details since strands and ties are not part of the augmented constraint diagram language. See [27] for various examples of statements that cannot be explicitly made when considering spider diagrams).

4.3 Reasoning

We will illustrate, by example, that sound reasoning ‘with a piece of syntax’ (in the case below, an edge) is dependent upon how that piece of syntax is used in the diagram. Ideally, a piece of syntax will behave consistently even when used in different contexts. In figure 13, $d_1$ expresses

$$PTC \land \forall x \in \text{OtherLang} \cup \text{English}\{x\}.all\text{Copies} = \text{OnLoan}$$

which is semantically equivalent to $d_2$ which, in turn, asserts

$$PTC \land \forall x \in \text{OtherLang}\{x\}.all\text{Copies} = \text{OnLoan}$$

$$\land \forall y \in \text{English}\{y\}.all\text{Copies} = \text{OnLoan}.$$ 

Notice that, to ‘obtain’ $d_2$ from $d_1$ the edge connecting the two asterisks is removed and an additional arrow is introduced. Furthermore, $d_1$ is also semantically equivalent to $d_3 \land d_4$; we say that the spider in $d_1$ has been split to give $d_3 \land d_4$. However, had the universal spiders in these four unitary diagrams
instead been existential spiders it would not be the case that \( d_1 \) is semantically equivalent to \( d_2 \) or to \( d_3 \land d_4 \). In this existential case, \( d_1 \) is semantically equivalent to \( d_3 \lor d_4 \). Thus, the edge is used to represent *disjunction* in one context, but *conjunction* in another.

The next example shows a reasoning step that is intuitive but not sound. In figure 14, \( d_1 \) is semantically equivalent to both \( d_2 \) and \( d_3 \). Given this, it seems reasonable to deduce that \( d_1 \) is also semantically equivalent to \( d_4 \) augmented with some appropriate reading tree (the only two valid trees for \( d_4 \) are shown above; notice that \( x \) and \( y \) are no longer independent in \( d_4 \)). However, choosing to augment \( d_4 \) with either of these valid trees does not give a diagram semantically equivalent to \( d_1 \); if we augment \( d_4 \) with the tree on the left then any interpretation in which *Film* represents the empty set is a model for \( d_4 \) but not necessarily a model for \( d_1 \). Intuitive reasoning can create dependencies between spiders which can potentially lead to errors in reasoning.

### 5 Introducing Generalized Constraint Diagrams

To summarize the discussions so far, the individual pieces of constraint diagram syntax are well-matched to meaning but the way in which they can be combined to make statements can lead to negative syntax interactions. Moreover, there are statements (some of which are relatively simple) that cannot be explicitly made by constraint diagrams; of these statements it is not always clear whether they are even expressible up to semantic equivalence.

In some of the examples we considered, the counter-intuitive features may well be linked to universal quantification over potentially empty sets (for example, figures 10 and 14). A possible reason for this link is because when a universal
spider, \( s \), has domain \( S \), this gives rise to a partial statement of the form \( \forall s (S(s) \Rightarrow ...) \) with the consequent subsequently derived (possibly implicitly) from the diagram and the reading tree. However, an ‘implicit’ consequent may itself (or part of it) make a statement that is visually apparent to the reader. As an example, in figure 10 the righthand diagram expresses

\[
\forall x \ (Film(x) \Rightarrow (\{x\}.leadActor \subseteq Actor \land \exists y \ Actor = \{y\})) \quad (*)
\]

where \( \exists y \ Actor = \{y\} \) is visually apparent but the fact that it forms part of a consequent must be derived from the diagram and the reading tree. By contrast, in \((*)\), \( \exists y \ Actor = \{y\} \) is obviously within the scope of the universal quantifier because the sentence is read in a linear and prescriptive way. We note that an alternative reading tree for this diagram, namely \( \bullet y \rightarrow \bullet x \), would not yield \( \exists y \ Actor = \{y\} \) as a part of a consequent even though it forms part of the reading.

In general, antecedents may also need to be derived from the diagram and the reading tree (for example, when a spider domain differs from the habitat). We observe that some work must be performed before one can read unitary diagrams due to their inexplicit nature whereas in the sentential case one can simply read from left to right.

A further example can be seen in figure 4. The sets over which the existential spiders quantify is determined, in part, by the order in which the quantifiers are read (given by a reading tree). Two valid trees are on the left of the diagram. The top most tree gives rise to the reading

\[
\exists q \left( \begin{array}{c}
Stock(q) \land \exists r \left( \begin{array}{c}
Stock(r) \land r \neq q \land \forall p \ (Film(p) \Rightarrow \\
(q, r, \exists s \{s\} = Stock - \{p\.allCopies\})
\end{array}\right)\end{array}\right).
\]

Using this tree, the diagram tells us that there are at least two elements in \( Stock \) (those represented by \( q \) and \( r \)) as well as many other things. Another valid, but only slightly different, tree is \( \bullet q \rightarrow \bullet p \rightarrow \bullet r \rightarrow \bullet s \) which gives rise to

\[
\exists q \left( \begin{array}{c}
Stock(q) \land \forall p \left( \begin{array}{c}
Film(p) \Rightarrow (\exists r \ (Stock(r) \land r \neq q \land \\
q, r \in \{p\.allCopies\}) \subseteq Stock \land \exists s \{s\} = Stock - \{p\.allCopies\})
\end{array}\right)\end{array}\right).
\]

which is rather different to the reading derived from the first tree and only tells us that there is necessarily one element in \( Stock \) (that represented by \( q \)), for example. The two given readings of the unitary diagram are semantically rather different yet there is only a small difference in the reading trees. By
contrast, the symbolic sentences have big differences between them. In the case of the diagram, one has to derive how to read the pieces of syntax by considering the diagram in conjunction with a valid tree. The reading tree provides the order in which to read the quantifiers and which other quantifiers are in their scope, but how to read the diagram and reading tree together is implicit and must be derived by the reader.

Thus, the question arises ‘can we improve the constraint diagram language?’ We propose a generalized version of constraint diagrams that utilizes most of the existing syntax, removes the need for reading trees and, we conjecture, is more expressive than the original notation; we prove that it is at least as expressive. The proposed generalization removes the need for reading trees to accompany the diagrams by making the quantifier reading order and scope explicit in the diagram itself. This can be achieved by viewing the diagram as a sequence of images (rather like watching someone draw the diagram) as opposed to a finished artifact with an accompanying tree. Such a sequence is analogous to the building sequence defined in [9] which is derived from a unitary diagram, $d$, and a reading tree in order to extract the formal meaning of $d$. However, this change alone would not overcome all of the aforementioned counter-intuitive features, in particular those covered in sections 4.2 and 4.3.

Instead, we will view a constraint diagram to be a partial order on a collection of images, extending the idea of using a sequence. Figure 15 illustrates the type of structure that might be present in a generalized constraint diagram, where the boxes are unitary diagrams. In general, each non-root unitary diagram in such a partial order will be a copy of its immediate ancestor with some changes to the syntax. Arrows connect the boxes, with those of the form $\downarrow\leftarrow$ indicating conjunction whereas those of the form $\downarrow\rightarrow$ indicate disjunction (adopting the convention that the vertical line represents disjunction); this will be more fully explained in the examples below. Disjunction and conjunction connectives of this form were also used in [16]. We exploit the associative properties of the disjunction and conjunction connectives by allowing the use of connecting arrows with many targets. Thus, a conjunction arrow with $n$ targets can be viewed symbolically as meaning $\bigwedge_{1 \leq i \leq n}$, for example. Moreover, arrows of the form $\rightarrow$ can be taken to represent either conjunction or disjunction, since they can be viewed as of the form $\bigwedge_{1 \leq i \leq 1}$ or $\bigvee_{1 \leq i \leq 1}$. The partial ordering will tell us when to read a piece of syntax (i.e. when it appears in the partial order) and (probably) allow us to increase the expressiveness of the language. Further modifications will be presented via examples.

There is a generalized constraint diagram in figure 16. The first (root) unitary diagram contains syntax which expresses

$$\forall x \in \text{Member } \{ x \}. \text{canBorrow} \subseteq \text{Film};$$
here the universal spider quantifies over its habitat. The second rectangle includes additional syntax, expressing that every element of \( \{ x \}.can\text{Borrow} \) is available to \( x \). Thus, this generalized constraint diagram is equivalent to that in figure 9 but the problem of domain differing from habitat is overcome. Moreover, the unhelpful symmetry present in figure 9 is lost in figure 16; here, the universal spider inside \textit{Member} occurs twice but the other universal spider only appears once.

Recall that the diagrams in figure 10 illustrated that the meaning of the shading depended on the other components in the diagram and this meaning had to be derived from the diagram and the reading tree. The diagram in figure 17 and the righthand diagram in figure 10 assert

\[
\forall x (\text{Film}(x) \Rightarrow (\{ x \}.\text{leadActor} \subseteq \text{Actor} \land \exists y \text{Actor} = \{ y \})).
\]

Since the shading appears in the sequence (figure 17) after the universal spider, we have explicit visual information that the meaning of the shading is dependent on that spider (this information was only implicit in figure 10).
that statements like
\[ \forall x \in \text{Film}\{x\}.\text{leadActor} \neq \emptyset \lor \{x\}.\text{allActors} = \emptyset \]
cannot be expressed by constraint diagrams. The generalized diagram in figure 18 easily makes this statement: the left hand box informs us that we are talking about all films (i.e. \( \forall x \in \text{Film} \)), then the subsequent diagrams give us \( \{x\}.\text{leadActor} \neq \emptyset \) (the top diagram) or \( \{x\}.\text{allActors} = \emptyset \) (the bottom diagram).

![Fig. 18. Fixing \( \forall \) and \( \lor \) issues.](image)

As illustrated in figure 13, how we reason with edges in the constraint diagram language is influenced by how those edges are used. In other diagrammatic systems, edges are typically used to make disjunctive statements (e.g. [26,32]). In figure 13, interpreting edges disjunctively would mean \( d_1 \) is semantically equivalent to \( d_2 \lor d_3 \) but not \( d_2 \), regardless of whether the spiders are all existential or all universal. In both cases, \( d_1 \) would not be semantically equivalent to \( d_2 \). We conjecture that always interpreting edges disjunctively would not decrease the expressive power of constraint diagrams. In generalized constraint diagrams, we will interpret edges disjunctively. Furthermore, for arrows sourced on spiders, we make the node from which the arrow emanates semantically important. In figure 19, the left hand diagram yields the partial statement \( \forall x \in \text{OtherLang} \cdots \lor \forall x \in \text{English} \cdots \). The right hand diagram completes the statement, with the whole diagram expressing
\[ \forall x \in \text{OtherLang}\{x\}.\text{allCopies} = \text{OnLoan} \lor \forall x \in \text{English}\{x\}.\text{allCopies} = \text{OnLoan} \]
as in the disjunction of \( d_3 \) and \( d_4 \) in figure 13, as opposed to the conjunction in the non-generalized case. Changing the spider type corresponds to replacing the \( \forall s \) by \( \exists s \) in the expression above.

![Fig. 19. Fixing reasoning with edges.](image)
Another issue with constraint diagrams is that intuitive reasoning can be unsound. In figure 14, we could add two zones individually, but adding both did not create a diagram semantically equivalent to the premise. This was because $x$ and $y$ were read independently in $d_1$, but adding both zones made them dependent. Consider now $d_5$ in figure 20. Here, the diagram forces one universal spider (that inside Member) to be inside the scope of the other. In this case, $d_5$ is semantically equivalent to both $d_6$ and $d_7$, so both zones can be added.

Finally, we previously noted that unitary constraint diagrams ensure distinct spiders to denote distinct elements. This means that, within the scope of a quantifier (i.e. a spider), all of the elements referred to explicitly are distinct. Figure 21 illustrates that this is not necessarily the case for generalized constraint diagrams. The diagram expresses (amongst other things)

$$\exists m \in \text{Member}(\exists f \in \text{Film canBorrow}(m, f) \land \exists g \in \text{Film hasBorrowed}(m, g))$$

but does not assert $f \neq g$. In a generalized unitary diagram, whilst spiders within any given rectangle do denote distinct elements, we have the flexibility to not enforce distinctness within the scope of quantifiers.
6 A Design Philosophy

Having introduced generalized constraint diagrams via a series of examples, linking back to the counter-intuitive features exhibited by constraint diagrams, we now proceed to outline the design principles that have been followed when producing the generalized notation. A priority is to ensure that generalized constraint diagrams are well-matched to their meaning. This explains our retention of Euler diagrams as the underlying notation, since they have clear qualities in this regard. We want to ensure that, when enriching Euler diagrams, semantic relations can be mirrored by syntactic relations. We now expose other considerations that informed the language design, inspired by [10,17] and the problems we have identified with constraint diagrams. In particular, our language design was strongly influenced by the following principles:

1. **Well-matchedness Principle**: require that syntactic relations mirror semantic relations.
2. **Explicitness Principle**: make the informational content explicit, not implicit.
3. **Interpretation Principle**: ensure that the semantics assigned to each piece of syntax are independent of context.
4. **Expressiveness Principle**: allow statements to be made naturally within the language.
5. **Construction Principle**: impose only the necessary restrictions on what constitutes a well-formed statement (in our case, a diagram).

There may arise situations where these design principles conflict, so there are no hard and fast rules that say we must completely satisfy them all. In general, the principles need to be balanced to produce a notation that finds a reasonable trade-off between them.
Generalized constraint diagrams retain the syntactic components of constraint diagrams and, in addition, use connectives between diagrams to allow partial orders of diagrams to be constructed. Each diagram in the partial order modifies the previous diagram and, in this section, we identify a necessary and sufficient list of relationships that must hold between a diagram and its ancestors in a partial order. The only constraints placed on the partial orders arise from semantic considerations in that they prevent ambiguous diagrams from being constructed; the semantics will be specified formally in the next section. The only way in which ambiguity can be introduced is through quantification. For example, if we modify a diagram by adding both an existential spider and a universal spider then we do not know in which order to read them, resulting in ambiguity. For example, the diagram in figure 22 adds both a universal and an existential spider to the first (root) diagram in a single step. Consequently, we do not know in which order to read the spiders, resulting in an ambiguous situation.

Our definition of the syntax follows the style of that in [9], where unitary diagrams are defined at an abstract level. The contours in our language will be drawn from two sets: $GC$ is the set of given contours and $DC$ is the set of derived contours. Further, the set $DC$ is countably infinite. At the abstract level, we identify given contours with their labels. In other words, the set $GC$ contains precisely the labels that we use for given contours at the drawn diagram (concrete) level. Derived contours, at the concrete level, do not have labels. Thus, at the abstract level, elements of $DC$ correspond to derived contours at the concrete level. The set $AL$ contains all of the arrow labels. We further define the sets $ES$ and $US$ which contain countably infinitely many existential spiders and universal spiders respectively. We assume that these five sets are pairwise disjoint. Before we define a unitary diagram, we first make precise various required concepts.

**Definition 7.1** A zone is an ordered pair $(in, out)$ where $in \subseteq GC \cup DC$ and $out \subseteq (GC \cup DC) - in$ and both $in$ and $out$ are finite.

As an example, in figure 23, there are three zones:
(1) that inside $A$ but outside $B$: ($\{A\}, \{B\}$),
(2) that inside $B$ but outside $A$: ($\{B\}, \{A\}$), and
(3) that outside both $A$ and $B$: ($\emptyset, \{A, B\}$).

The zone ($\{A, B\}, \emptyset$) is not present in the diagram since there is no region inside both $A$ and $B$.

![Diagram](image_url)

Fig. 23. Formalizing zones and arrows.

**Definition 7.2** An arrow end is either a spider together with a zone or a contour; in the former case the arrow end is of the form $(sp, z)$ where $z$ is a zone and $sp$ is a spider. An arrow is an ordered triple $(s, l, t)$ where $s$ is an arrow end, called the source, $l$ is an arrow label and $t$ is an arrow end, called the target.

In figure 23, the arrow is $((s, (\{A\}, \{B\})), f, B)$; the pair $(s, (\{A\}, \{B\}))$ tells us that the arrow is sourced on the foot of $s$ located in the zone $\{A\}$.

**Definition 7.3** A generalized unitary diagram is a tuple,

$$d = (Z, ShZ, ES, US, hab, A)$$

which satisfies the following.

(1) $Z = Z(d)$ is a finite set of zones such that each pair of zones, $(in_1, out_1)$, $(in_2, out_2)$, in $Z(d)$ satisfies $in_1 \cup out_1 = in_2 \cup out_2$.

(2) $ShZ = ShZ(d)$ is a finite set of shaded zones such that $ShZ(d) \subseteq Z(d)$

(3) $ES = ES(d)$ is a finite set of existential spiders.

(4) $US = US(d)$ is a finite set of universal spiders.

(5) $hab = hab_d$ is a function, $hab_d: ES(d) \cup US(d) \rightarrow \mathcal{P}Z(d) - \{\emptyset\}$ which returns the habitat of each spider.

(6) $A = A(d)$ is a set of arrows such that each arrow, $(s, l, t)$, in $A(d)$ has ends which are in the diagram. More formally, $s$ and $t$ are elements of one of the following sets:

(a) $in \cup out$ where $(in, out) \in Z(d)$, or
(b) $\{(s, z) : s \in ES(d) \cup US(d) \land z \in hab_d(s)\}$.

For a generalized unitary diagram, $d$, we define:
(1) $C(d)$ to be the set of contours in $d$, given by $C(d) = \text{in} \cup \text{out}$ where $(\text{in}, \text{out}) \in Z(d)$,
(2) $S(d)$ to be the set of spiders in $d$, given by $S(d) = ES(d) \cup US(d)$, and
(3) $DC(d)$ to be the set of derived contours in $d$, given by $DC(d) = DC \cap C(d)$.

The set of all generalized unitary diagrams is denoted $UD$.

The diagram in figure 23 has the following abstract syntax:

(1) $Z(d) = \{((\{A\}, \{B\}), (\{B\}, \{A\}), (\emptyset, \{A, B\}))\},$
(2) $ShZ(d) = \emptyset,$
(3) $ES(d) = \{s\},$
(4) $US(d) = \emptyset,$
(5) $habd(s) = \{((\{A\}, \{B\}), (\emptyset, \{A, B\}))\},$ and
(6) $A(d) = \{((s, (\{A\}, \{B\})), f, B)\}.$

At the abstract level, we will define a generalized constraint diagram to be a labelled, directed, bipartite tree, in which the vertices from one of the two partite sets are labelled by unitary diagrams. The vertices in the other partite set are labelled by the connectives $\lor$ (corresponding to the visualization $\rightarrow\rightarrow\rightarrow$) or $\land$ (corresponding to $\rightarrow\uparrow\rightarrow$); we note that vertices in this set with out-degree equal to one can be labelled with either $\land$ or $\lor$ without any impact on the semantics.

Definition 7.4 A diagram-labelled tree is a (vertex) labelled, directed bipartite tree, $T$, with partite sets $V_1$ and $V_2$ such that:

(1) the vertices in $V_1$ are each labelled by a generalized unitary diagram,
(2) the vertices in $V_2$ are labelled by either $\land$ or $\lor$,
(3) there is exactly one root vertex in $T$ which is in the set $V_1$,
(4) every vertex in $V_1$, other than the root vertex, has in-degree equal to 1 (vertices labelled by unitary diagrams are the target of an edge),
(5) every vertex in $V_1$ has out-degree equal to at most 1 (vertices that are labelled by unitary diagrams are the source of at most one edge),
(6) every vertex in $V_2$ has in-degree equal to 1 and non-zero out-degree.

Vertices with out-degree equal to zero are called leaves of $T$.

The diagram in figure 15 constitutes a diagram-labelled tree whereas that in figure 24 does not for a variety of reasons (note that we have emphasized the vertices implicitly labelled by connectives in this figure for illustration purposes): it is not a tree, it contains two diagram-labelled vertices with in-degree 0, two diagram-labelled vertices have out-degree 2, two diagram-labelled vertices have in-degree 2, and connective-labelled vertex has in-degree equal to two.
Definition 7.5 Let \( T = (V(T), E(T)) \) be a diagram-labelled tree and let \( v_1 \) and \( v_2 \) be vertices of \( T \) that are labelled by unitary diagrams. The vertex \( v_1 \) is an **immediate ancestor** of \( v_2 \) if there is a path of length two from \( v_1 \) to \( v_2 \) (such a path passes through exactly one vertex and this vertex is labelled by \( \land \) or \( \lor \)); in such a case, \( v_2 \) is called an **immediate descendant** of \( v_1 \). The **ancestor** relation is the transitive closure of immediate ancestor.

We note that immediate ancestors are unique but a vertex can have many immediate descendants.

We will blur the distinction between vertices and their labels where it is convenient for us to do so. We now proceed to define a collection of terms that allow us to identify what constitutes a generalized constraint diagram, using the notion of a diagram-labelled tree. In particular, we want to identify relationships between diagrams that allow them to be consecutive labels on a tree (in the sense of immediate ancestor and descendant) and which diagrams are allowed to label roots.

Clearly, any unitary diagram, \( d \), which is used to label a root cannot contain more than one universal spider or it will be ambiguous. If \( d \) does contain a universal spider then it cannot contain any existential spiders (\( \forall \) and \( \exists \) do not commute). There is no constraint on the number of existential spiders that can be present in \( d \) other than that just given. The diagram in figure 25 cannot be a root since it contains two universal spiders; the order in which they are read, and their scope, impacts the meaning of the diagram.

Definition 7.6 Let \( d \) be a generalized unitary diagram. We say that \( d \) is a **root diagram** whenever
(1) $d$ contains no universal spiders or
(2) $d$ contains exactly one universal spider and no existential spiders.

Similarly, a diagram in a partial order must satisfy some constraint with regard to the spiders it possesses, in relation to the spiders present in any of its ancestors; we cannot add new spiders arbitrarily for the same reasons as given for root diagrams.

**Definition 7.7** Let $d_0, \ldots, d_n$ be generalized unitary diagrams. If

(1) the universal spiders in $d_n$ are all contained in the other diagrams:

$$US(d_n) \subseteq \bigcup_{0 \leq i \leq n-1} US(d_i)$$

or

(2) there is a unique new universal spider in $d_n$ and no new existential spiders:

$$|US(d_n) - \bigcup_{0 \leq i \leq n-1} US(d_i)| = 1 \text{ and } ES(d_n) \subseteq \bigcup_{0 \leq i \leq n-1} ES(d_i).$$

then $d_n$ is called a **valid descendant** of $d_0, \ldots, d_{n-1}$.

**Definition 7.8** Let $T$ be a diagram-labelled tree. We say that $T$ is a **generalized constraint diagram** provided

(1) the root vertex is labelled by a root diagram, and
(2) each vertex, $v$, other than the root vertex, is labelled by
   (a) a valid descendant of the diagrams which label the ancestors of $v$ or
   (b) a logical connective.

**8 The Semantics of Generalized Constraint Diagrams**

We will define the semantics in a recursive style, since the partial ordering seen in our diagrams reflects the semantic construction. We will define, for each unitary diagram, a formula to which it corresponds. These formulae, together with the insertion of quantifiers and connectives at appropriate points, will form sentences corresponding to diagrams. First, we define interpretations, analogous to structures in predicate logic, which map the given contours and arrow labels to relations on some universal set.

**Definition 8.1** An **interpretation** is a triple, $(U, \Psi, \Phi)$, where $U$ is a non-empty set, $\Psi: GC \rightarrow \mathcal{P}U$ is a function that maps given contours to subsets of $U$ and $\Phi: \mathcal{AL} \rightarrow \mathcal{P}(U \times U)$ maps arrow labels to binary relations on $U$ [28].
In what follows, we assume that $\Psi$ has been extended to interpret derived contours as subsets of $U$. We will construct such extensions of $\Psi$ when giving the formal semantics of a diagram but for our current purposes it is merely sufficient to assume such an extension has been provided. As with previous work [28], it is convenient to further extend $\Psi$ to include the interpretations of zones and sets of zones; the images of the zones (and sets of them) are completely determined by the $U$ and the images of the contours. Given an extension of $\Psi$ to $\Psi: \mathcal{GC} \cup \mathcal{DC} \rightarrow \mathcal{P}U$,

(1) for each zone, $(in, out)$, we define
\[
\Psi(in, out) = \bigcap_{c \in in} \Psi(c) \cap \bigcap_{c \in out} (U - \Psi(c))
\]

(2) for each set of zones, $Z$, we define
\[
\Psi(Z) = \bigcup_{z \in Z} \Psi(z).
\]

For unitary diagrams, $d$, we will specify a collection of conditions (given in definition 8.4) whose conjunction captures the meaning of $d$ when $d$ is viewed as a formula not containing quantifiers. These conditions correspond to the syntactic components in $d$ and their relationships with each other. To specify the meaning assigned to arrows, we must consider their ends. There are two types of arrow ends: contours and spider feet. Contours represent sets whereas spiders represent elements so we need to consider several cases, reflecting this difference. For example, suppose an arrow is sourced on a given contour, $A$, and has label $l$. If the target is a contour, $C$, then the arrow asserts that $A.l = C$. However, if the target is instead a spider foot, say $(s, z)$, then the arrow asserts that $A.l = \{s\}$ provided $s$ represents an element in the set represented by $z$. We have to ‘turn’ $s$ into a set since it is ambiguous to overload the notation and write $A.l = s$ (we formally define $A.l$ later). We also have to take into account the location of the element represented by $s$, since the arrow is sourced on the foot, rather than $s$; this location is taken to be the zone that represents the set which contains the represented element.

**Definition 8.2** Let $d$ be a generalized unitary diagram. We define a four-way partition of the arrow set $A(d)$ as follows.

1. $Acc(d) = \{(s, l, t) \in A(d) : s, t \in C(d)\}$, the set of arrows sourced and targeted on contours.
2. $Asc(d) = \{(s, l, t) \in A(d) : s \in S(d) \times Z(d) \land t \in C(d)\}$, the set of arrows sourced on spider feet and targeted on contours.
3. $Acs(d) = \{(s, l, t) \in A(d) : s \in C(d) \land t \in S(d) \times Z(d)\}$, the set of arrows sourced on contours and targeted on spider feet.
4. $Ass(d) = \{(s, l, t) \in A(d) : s, t \in S(d) \times Z(d)\}$, the set of arrows sourced and targeted on spider feet.
Definition 8.3 Let \( U \) be a set, let \( R \) be a binary relation on \( U \) and let \( A \) be a subset of \( U \). We define the \textit{image} of \( R \) when the domain is restricted to \( A \), denoted \( A.R \), to be \( A.R = \{ y \in U : \exists x \in A(x,y) \in R \} \).

Each spider, \( s \), gives rise to an expression of the form \( Qx \in S \) where \( Q \) is a quantifier whose type is determined by \( s \) and \( S \) is a set determined by the placement of \( s \) in a diagram. We define \( \mathcal{V} \) to be a countably infinite set of variables and further define a bijection, \( sv: US \cup ES \to \mathcal{V} \); this bijection is taken to be fixed and will be used in our construction of the semantics.

Definition 8.4 Let \( d \) be a generalized unitary diagram and let \( (U, \Psi, \Phi) \) be an interpretation.

1. The \textit{plane tiling condition} for \( d \) asserts that the union of the sets represented by the zones is the universal set: \( \Psi(Z) = U \).
2. The \textit{shaded zones condition} for \( d \) asserts that all of the elements in the sets represented by shaded zones are represented by spiders:
   \[
   \bigwedge_{z \in S(d)} \Psi(z) \subseteq im(sv)|_{S(d)}
   \]
   where \( im(sv)|_{S(d)} \) is the image of the function \( sv \) with the domain restricted to \( S(d) \).
3. The \textit{spiders’ habitats condition} for \( d \) asserts that the elements represented by the spiders are in the sets represented by their habitats:
   \[
   \bigwedge_{s \in S(d)} sv(s) \in \Psi(\text{hab}_d(s)).
   \]
4. The \textit{spiders distinctness condition} for \( d \) asserts that distinct spiders represent distinct elements:
   \[
   \bigwedge_{s_1, s_2 \in S(d) \land s_1 \neq s_2} sv(s_1) \neq sv(s_2).
   \]
5. The \textit{arrows condition} for \( d \) asserts that, for each arrow, the set represented by the arrow’s target is the image of the relation represented by the label when the domain of that relation is restricted to the set represented by the source. To specify this condition formally, we take the conjunction of four assertions, relating to the four-way partition of the arrow set given above.
   \[
   \text{Assertion 1:} \quad \bigwedge_{(s,l,t) \in Acc(d)} \Psi(s).\Phi(l) = \Psi(t)
   \]
   \[
   \text{Assertion 2:} \quad \bigwedge_{((sp,z),l,t) \in Asc(d)} \,(sv(sp) \in \Psi(z) \Rightarrow \{sv(sp)\}.\Phi(l) = \Psi(t))
   \]
   \[
   \text{Assertion 3:} \quad \bigwedge_{(s,l,(sp,z)) \in Acs(d)} \,(sv(sp) \in \Psi(z) \Rightarrow \Psi(s).\Phi(l) = \{sv(sp)\})
   \]
Assertion 4:

\[ \bigwedge_{((sp_1,z_1),l,(sp_2,z_2)) \in As(d)} ((sv(sp_1) \in \Psi(z_1) \land sv(sp_2) \in \Psi(z_2)) \Rightarrow \Psi(sv(sp_1)).\Phi(l) = \{sv(sp_2)\}). \]

The conjunction of the above five conditions is called the formula for \( d \), denoted \( \text{form}(d) \).

Fig. 26. The formula of a unitary diagram.

To illustrate, the diagram in figure 26 has \( \text{form}(d) \) given by the conjunction of the following conditions.

1. Plane tiling condition: \( \Psi(\{(\{A\}, \{B\}), (\{B\}, \{A\}), (\emptyset, \{A, B\})\}) = U \).
2. Shaded zones condition: \( \top \), where \( \top \) denotes true.
3. Spiders’ habitats condition:
   \[ x_1 \in \Psi(\{(\{A\}, \{B\}), (\emptyset, \{A, B\})\}) \land x_2 \in \Psi(\{(\{A\}, \{B\})\}) \]
   where \( s_1 \) and \( s_2 \) are the spiders in \( d \), \( sv(s_1) = x_1 \) and \( sv(s_2) = x_2 \).
4. Spiders distinctness condition: \( x_1 \neq x_2 \).
5. Arrows condition: \( x_1 \in \Psi(\emptyset, \{A, B\}) \Rightarrow \{x_1\}.\Phi(f) = \Psi(B) \).

When all zones are present in a unitary diagram \( d \) given the contours present, the plane tiling condition is true [11]; thus we may choose not to write down the plane tiling condition in such circumstances. Equivalent to the plane tiling condition is the missing zones condition which asserts that the sets represented by the zones which are not present, given the contours in the diagram, all represent the empty set. The plane tiling condition and the missing zones condition are interchangeable in the formula for a unitary diagram.

We take the convention that spiders take precedence when interpreting the diagrams. For example, given a root diagram, \( d \), this means that we prefix \( \text{form}(d) \) by one quantifier for each spider in \( d \), including information as to which set we are quantifying over. This set is broken down into the parts represented by the zones in which the spider has feet, illustrated below. For diagrams that are not roots, we only need to write down quantifiers for any ‘new’ spiders.

Definition 8.5 Let \( d \) be a generalized unitary diagram. Let \( s \in S(d) \). The
set of quantification expressions for \( s \) in \( d \), denoted \( Q(s, d) \), is given by

\[
\{ \text{Quant } sv(s) \in \Psi(z) : z \in \text{hab}_d(s) \}
\]

where \( \text{Quant } = \exists \) if \( s \) is existential and \( \text{Quant } = \forall \) otherwise. Let \( S \) be a set of spiders in \( d \) (that is, \( S \subseteq S(d) \)) and \( f : S(d) \to \{1, \ldots, |S|\} \) be a function that gives rise to a total order on the spiders in \( S \). The set of quantification expressions for \( S \) given \( f \), denoted \( Q(S, d) \), is given by

\[
\{ \text{exp}_1 \ldots \text{exp}_{|S|} : \text{exp}_i \in Q(s, d) \text{ where } s \in S \text{ and } f(s) = i \}.
\]

For example, in figure 26, the two footed spider, \( s_1 \), has quantification expressions \( \exists x_1 \in \Psi(\{A\}, \{B\}) \) and \( \exists x_1 \in \Psi(\emptyset, \{A, B\}) \). The other spider, \( s_2 \) has quantification expression \( \exists x_2 \in \Psi(\{A\}, \{B\}) \). For \( S(d) \), there are two possible total orders on the spiders, giving rise to two sets of quantification expressions:

\[
\{ \exists x_1 \in \Psi(\{A\}, \{B\}) \exists x_2 \in \Psi(\{A\}, \{B\}), \exists x_1 \in \Psi(\emptyset, \{A, B\}) \exists x_2 \in \Psi(\{A\}, \{B\}) \}
\]
and

\[
\{ \exists x_2 \in \Psi(\{A\}, \{B\}) \exists x_1 \in \Psi(\{A\}, \{B\}), \exists x_2 \in \Psi(\{A\}, \{B\}) \exists x_1 \in \Psi(\emptyset, \{A, B\}) \}.
\]

Taking \( d \) to be a root diagram, we use one of the sets of quantification expressions for \( S(d) \) to produce the meaning of \( d \). It does not matter which set we choose from a semantic perspective: the order in which we write the quantification expressions for the existential spiders does not change the meaning. That is, the choice of total ordering has no impact on the meaning of \( d \). So, \( d \) is taken to mean:

\[
\exists x_1 \in \Psi(\{A\}, \{B\}) \exists x_2 \in \Psi(\{A\}, \{B\}) \text{ form}(d) \lor \\
\exists x_1 \in \Psi(\emptyset, \{A, B\}) \exists x_2 \in \Psi(\{A\}, \{B\}) \text{ form}(d).
\]

When constructing the semantics of a diagram, we use the formulae for the labelling unitary diagrams and add quantifiers and connectives as appropriate. For the root diagram, all spiders give rise to a quantifier but, as we progress through the tree structure, we only wish to introduce quantifiers for the new spiders. The only situations when the total ordering on the spiders could impact the semantics occur when at least two new spiders are involved of which one is universal. However, root diagrams do not let this scenario arise. Moreover, for non-root unitary diagrams \( d \), the constraint placed on the ‘new’ spiders in \( d \) ensures that this scenario does not arise; for the semantics associated with \( d \), we only write down quantification expressions for the new spiders. Thus, when formalizing the semantics, we do not concern ourselves with the different choices of total orderings on the new spiders, but just assume the existence of such an ordering.
When forming quantification expressions for diagrams, we need to ensure that the sets over which we are quantifying are defined. When the zones only involve given contours this is easy: all of the given contours are mapped to sets in an interpretation. When derived contours are involved, the situation is not so obvious. For example, taking $d_1$ in figure 27 as a generalized diagram, without giving due consideration to the derived contour, $dc$, we might write down, as the semantics,

$$\exists s_1 \in \Psi(\{\{dc\}, \{A\}\}, (\emptyset, \{A, dc\})) \text{ form}(d_1).$$

However, we have not interpreted $dc$ as a set at the point of writing down the quantification expression; the set which $dc$ represents becomes apparent only when we assert $A.f = dc$ in the arrows condition. We take this into account in two ways. First, we describe the habitat of the spider, $s$, without reference to the uninterpreted derived contour. We can see that the spider’s habitat can be described as being outside $A$; the diagram $d_2$ is obtained from $d_1$ by removing $dc$ and gives the ‘new habitat’ of $s$. Secondly, within the scope of the quantifier arising from $s$, we assert the existence of an extension of $\Psi$ to interpret the derived contour $dc$. Our approach results in our semantics being second-order. In our example, the semantics of $d_1$ therefore are

$$\exists x_1 \in \Psi(\{(\emptyset, \{A\})\}) \exists \Psi: GC \cup DC(d_1) \rightarrow PU \text{ form}(d_1)$$

where $\Psi: GC \cup DC(d_1) \rightarrow PU$ extends $\Psi$ to interpret the derived contour in $d_1$.

**Definition 8.6** Let $d$ be a generalized diagram. Let $v$ be a vertex in $d$ and suppose that unitary diagram $d_i$ labels $v$ in $d$. We define the set of new spiders at $v$ in $d$, denoted $NewS(v, d)$, to be

$$NewS(v, d) = S(d_i) - \left( \bigcup_{d_j \in \text{Anc}(v, d)} S(d_j) \right)$$

where $\text{Anc}(v, d)$ is the set of unitary diagrams which label the ancestors of the vertex $v$ in $d$. Similarly, we define the set of new derived contours at $v$ in $d$, denoted $NewDC(v, d)$.

In the above definition, if $d_i$ labels the root of $d$ then $NewS(v, d) = S(d_i)$ and $NewDC(v, d) = DC(d_i)$ because $d_i$ has no ancestors.
Definition 8.7 Let \( d \) be a generalized unitary diagram and let \( DC \) be a set of derived contours. The underlying diagram of \( d \) given \( DC \), denoted \( d - DC \), has components given by

1. \( Z(d - DC) = \{(in - DC, out - DC) : (in, out) \in Z(d)\} \)
2. \( ShZ(d - DC) = \{(in - DC, out - DC) : (in, out) \in ShZ(d)\} \)
3. \( ES(d - DC) = ES(d) \)
4. \( US(d - DC) = US(d) \)
5. \( hab_{d-DC} : S(d - DC) \rightarrow PZ(d - DC) - \{\emptyset\} \) satisfies
   \[ hab_{d-DC}(s) = \{(in - DC, out - DC) : (in, out) \in hab_d(s)\} \]
6. \( A(d - DC) = \emptyset \)

In figure 27, \( d_2 \) is the underlying diagram of \( d_1 \) given \( DC(d_1) \). Intuitively, underlying diagrams are obtained by deleting all of the derived contours in \( DC \) along with all of the arrows.

We are now in a position to construct, for a generalized constraint diagram, a sentence which allows us to determine whether any given interpretation agrees with the meaning of the diagram; such interpretations are called models. We take a recursive approach, producing formulae for the leaves and building up an expression which yields a sentence for the generalized diagram.

Definition 8.8 Let \( d \) be a generalized constraint diagram and let \( v \) be a vertex in \( d \) with label unitary diagram \( d_i \). Let \((U, \Psi, \Phi)\) be an interpretation. We define the semantic formula for \( v \) in the context of \( d \), denoted \( SemForm(v, d) \), as follows. First, define

\[ Quant(v, d_i) = Q(NewS(v, d), d_i - NewDC(v, d)) \]

and

\[ Ext(v, d_i) = \exists \Psi : GC \cup NewDC(v, d) \rightarrow PU, \]

to simplify the representation of the formulae below.

(1) If \( v \) is a leaf vertex then \( SemForm(v, d) \) is the formula

\[ \bigvee_{QExp \in Quant(v, d)} QExp Ext(v, d_i) form(d_i). \]

(2) If \( v \) is a vertex that is not a leaf of \( d \), and the source of an edge labelled \( \lor \) then \( SemForm(v, d) \) is the formula

\[ \bigvee_{QExp \in Quant(v, d)} QExp Ext(v, d_i) (form(d_i) \land ( \bigvee_{d_j \in ImDec(v, d)} SemForm(v, d_j))) \]

where \( ImDec(v, d) \) is the set of diagrams that label the immediate descendants of \( v \) in \( d \).
(3) Otherwise \( v \) is a vertex that is not a leaf of \( d \), and the source of an edge labelled \( \land \) in which case \( \text{SemForm}(v, d) \) is the formula

\[
\bigvee_{\text{QExp} \in \text{Quant}(v,d_i)} \text{QExp Ext}(v, d_i) (\text{form}(d_i) \land \bigwedge_{d_j \in \text{ImDec}(v,d)} \text{SemForm}(v, d_j)).
\]

Fig. 28. Interpreting a generalized diagram.

For any generalized constraint diagram, the semantic formula of the root vertex is a sentence. To illustrate the process, the leaf node \( d_2 \) in figure 28 has \( \text{SemForm}(d_2) \) given by

\[
\exists x_2 \in \Psi(\{(\emptyset, \{A\})\}) (x_1 \in \Psi(\{\{A\}, \emptyset\})) \land x_2 \in \Psi(\{(\emptyset, \{A\})\}) \land x_1 \neq x_2 \land (x_1 \in \Psi(\{A\}, \emptyset) \Rightarrow \{x_1\}.\Phi(f) = \{x_2\})).
\]

The semantic formula for the root \( d_1 \), and therefore for the diagram \( d_1 \rightarrow d_2 \), is

\[
\forall x_1 \in \Psi(\{\{A\}, \emptyset\}) (x_1 \in \Psi(\{\{A\}, \emptyset\}) \land \text{SemForm}(d_2)).
\]

We observe that there may appear to be some redundancy in the above semantic formulae in that spider’s give rise to two very similar components: each spider’s quantification expression is similar to its contribution to the spiders’ habitats condition.

Fig. 29. Habitat information is not redundant.

In general, this information is not necessarily redundant since, for example, a spider’s habitat may change as the partial order evolves. For example, in figure 29, the habitat of the spider is \( \{(\emptyset, \emptyset)\} \) in the root diagram but changes to \( \{(\{A\}, \{dc\})\} \) in the leaf diagram, where \( dc \) denotes the derived contour. The semantic formula for \( d_1 \rightarrow d_2 \) is

\[
\exists s_1 \in \Psi(\{(\emptyset, \emptyset)\}) (s_1 \in \Psi(\{(\emptyset, \emptyset)\}) \land \text{SemForm}(d_2))
\]
where $\text{SemForm}(d_2)$ asserts the existence of $\Psi: \mathcal{GC} \cup \text{NewDC}(d_2) \rightarrow P U$ for which

$$\Psi(\{(\{dc\}, \emptyset), (\emptyset, \{dc\})\}) = U \land s_1 \in \Psi(\{(\emptyset, \{dc\})\}) \land (s_1 \in \Psi(\{(\emptyset, \{dc\})\}) \Rightarrow \{s_1\}, \Phi(l) = \Psi(dc))$$

holds.

**Definition 8.9** Let $d$ be a generalized diagram, let $v$ be the root vertex of $d$ and let $I = (U, \Psi, \Phi)$ be an interpretation. We say that $I$ satisfies $d$ if $\text{SemForm}(v, d)$ is true. If $I$ satisfies $d$ then we say that $I$ is a model for $d$.

We observed above that there is a choice as to how we construct $\text{SemForm}(v, d)$ where $v$ is the root vertex of $d$ precisely due to the need to totally order new spiders. However, as we argued, this non-uniqueness is not of semantic importance. In other words the notions of satisfaction and model are well-defined.

**Theorem 8.1** Let $d$ be a generalized constraint diagram. Then the choice of $\text{SemForm}(v, d)$ has no impact on the set of interpretations which model $d$.

**Proof**[Sketch] The proof proceeds by induction on number of descendants of each vertex labelled by a unitary diagram in $d$. Let $v_j$ be such a vertex and assume that, for all descendants, $v_i$, of $v_j$, any two choices of $\text{SemForm}(v_i, d)$ are semantically equivalent. Then, when constructing $\text{SemForm}(v_j, d)$, the only possible way of obtaining semantically different interpretations is through the choice of total ordering on the new spiders. If the diagram, $d_j$, labelling $v_j$ has new universal spiders then such a spider is unique (by the definition of a generalized diagram) and the result holds. If $d_j$ contains new existential spiders then, as argued above, such choices have no impact on the semantics because

$$\exists x \in S \exists y \in T \land F,$$

where $F$ is some formula, is shorthand for

$$\exists x (S(x) \land \exists y (T(y) \land F))$$

and $\exists$ distributes over $\land$ and is commutative. That is, we could instead write

$$\exists x \exists y (S(x) \land T(y) \land F)$$

by distributivity, which is equivalent to

$$\exists y \exists x (T(y) \land S(x) \land F)$$

by commutativity. Then we can easily obtain

$$\exists y (T(y) \land \exists x (S(x) \land F)).$$

Hence the result holds by induction. □
9 Meeting the Design Principles

Having formalized generalized constraint diagrams, we now include some discussions on how we have met the five design principles that we stated earlier in the paper. It seems appropriate to discuss first our design of the syntax. It should be apparent that the restrictions we have placed on what constitutes a generalized constraint diagram are only those necessary for being able to interpret the diagrams unambiguously, thus adhering to the construction principle. This allows a large degree of freedom in terms of how statements are constructed. From our experience, this has desirable consequences when attempting to reason with notations, since one is able to perform seemingly natural reasoning steps that might otherwise have not been possible due to an overly constrained syntax.

Such flexibility brings with it, however, the ability to create diagrams which are not so well matched to their meaning. Thus there is some onus placed on users of the notation to choose appropriate representations of their information that are well-matched. For example, it is possible to create diagrams where the first occurrence of a spider does not have its domain equal to its habitat, but such diagrams can always be replaced by other semantically diagrams where the domain and habitat are the same. The two semantically equivalent diagrams in figure 30 illustrate this type of situation. Both diagrams assert that there is an element, \( e \), in the universe (this is explicit in the righthand diagram only) that is related to itself under \( f \). In our opinion there is probably not too much difficulty associated with interpreting the lefthand diagram where the domain is different from the habitat because the diagram is very simple. The empirical study in [10] suggests that people have a preference for diagrams with domain and habitat coinciding and our new notation allows this to be achieved. Had we prevented diagrams with domain and habitat differing from being created, this would have conflicted with the construction principle.

![Fig. 30. Habitat vs domain considerations.](image)

Our new notation also exhibits other features that relate well with the results in [10]: we read diagrams from left to right and our interpretation principle means that we have defined the semantics in a modular way so that arrows can be read independently of one another. This means that users who prefer to ‘follow arrows’ will not derive an incorrect meaning by reading in this way.
We claim that the syntax allows the construction of diagrams that are well-matched, justified by the examples given in section 5 and the discussions in section 3 on the individual pieces of syntax and that we have followed the well-matchedness principle.

It is easy to see that we have achieved the interpretation principle. For unitary diagrams, their semantic formula considers individual pieces of syntax and assigns a meaning to them. Since we have taken these assigned meanings in conjunction, we are able to look at a piece of syntax within a unitary diagram and know what that piece means.

The way in which the diagrams are constructed, being partial orders, and the way in which the semantics have been defined, ensures that the informational content can now be made explicit. The issues to do with, for example, having to extract a consequent which is only implicit from the visualization (as discussed in section 5) no longer occurs: scoping information is always explicit since spiders quantify over all of the information provided by the unitary diagram, $d$, in which they first occur and all of the unitary diagrams which are descendants of $d$ in the partial order. We can explicitly see which pieces of syntax fall within the scope of which quantifiers, unlike in the non-generalized case. We have adhered to the explicitness principle.

With regard to the expressiveness principle, the examples given in section 5 go some way towards justifying that we have met this principle. We certainly believe that many statements are more naturally made with generalized constraint diagrams than with constraint diagrams. However, it is likely that extensive user studies will be required to properly evaluate the relative strengths and weaknesses of the two notations in this regard. From a theoretical perspective, a partial answer can be achieved by proving that generalized constraint diagrams are at least as expressive as constraint diagrams, which we show to be the case in the next section. It is only possible to evaluate the extent to which we have achieved this principle now that we have formalized the syntax and semantics of the generalized notation.

10 Expressiveness

We will now proceed to show that generalized constraint diagrams are capable of expressing a specified fragment of first order logic. Moreover, we also show that this fragment is at least as expressive as constraint diagrams. From this, it follows that generalized constraint diagrams are at least as expressive as constraint diagrams. Recall that we have $\mathcal{V}$ as a set of variables; we use this set in our definitions below. We identify given contours with monadic predicate symbols (i.e. monadic predicate symbols are drawn from the set $\mathcal{GC}$)
and identify arrow labels with binary predicate symbols (i.e. binary predicate symbols are drawn from the set $\mathcal{AL}$). The particular fragment of first order language that is of interest contains:

(1) **Atomic Formulae**
(a) $(x = y)$ where $x, y \in \mathcal{V}$,
(b) $P(x)$ where $x \in \mathcal{V}$ and $P \in \mathcal{GC}$, and
(c) $R(x, y)$ where $x, y \in \mathcal{V}$ and $R \in \mathcal{AL}$.

(2) **Formulae**
(a) atomic formulae are formulae,
(b) if $F_1$ and $F_2$ are formulae then so are $\neg F_1$, $(F_1 \lor F_2)$, $(F_1 \land F_2)$, and $(F_1 \Rightarrow F_2)$, and
(c) If $F_1$ is a formula and $x$ is a variable then $\forall x F_1$ and $\exists x F_1$ are formulae.

Moreover, we make use of the following three well-known results.

**Theorem 10.1** The fragment of first order logic specified above is undecidable [2].

**Theorem 10.2** Let $F$ be a formula drawn from the fragment of first order logic specified above. Then $F$ is semantically equivalent to its universal closure [20].

Consequently, to establish our expressiveness results, we assume that we are dealing with sentences only.

**Theorem 10.3** Let $S$ be a sentence drawn from the fragment of first order logic specified above. Then $S$ is semantically equivalent to a sentence in prenex normal form, say $Q_1 x_1 \ldots Q_n x_n F$, where $F$ is quantifier free and in disjunctive normal form, each $Q_i$ is a quantifier and each $x_i$ is a variable.

For the above theorem, see [3] for the result that says any sentence can be transformed into prenex normal form, say $Q_1 x_1 \ldots Q_n x_n G$ where $G$ is quantifier free, and then apply standard logical equivalences to transform $G$ into disjunctive normal form.

We will take a sentence, $S$, in this form and show how to convert it into generalized constraint diagram. The translation process will involve converting the literals in $S$ into (roughly speaking) unitary diagrams that we use to label a generalized constraint diagram. We note that the translation process we give is not intended to produce an elegant diagram, only one with the same semantics as the original sentence; finding an elegant diagram as a result of the translation is a challenging task in the general case because of the rather different syntax possessed by the two languages. We merely aim to show that a translation exists to establish an expressiveness result from a theoretical perspective.
Suppose the sentence, $S$, that we wish to translate is of the form $\exists x_1 \forall x_2 \exists x_3 F$. Then, regardless of $F$, we construct a unitary diagram for each quantifier as shown in figure 31.

Since $F$ is in disjunctive normal form, we will construct a generalized diagram, $d$, that contains one disjunction connective and, for each conjunct, one conjunction connective. The general structure can be seen in figure 32, where each $s_i$ arises from quantifier $Q_i$. We will now work through a sufficiently rich
example to illustrate most of the translation process. In particular, we note that literals in our first order language are of the form $P(x)$, $R(x, y)$, $x = y$ (the positive literals) and their negations. A translation of the sentence

$$\forall x_1 \exists x_2 \exists x_3 \forall x_4 ((P(x_1) \land \neg R(x_4, x_2) \land \neg (x_1 = x_3)) \lor (\neg P(x_1) \land R(x_1, x_2)))$$

can be seen in figure 33. For ease of matching diagram parts with the symbolic sentence given, we have labelled each spider by the variable associated with the quantifier from which it arises.

The literal $P(x_1)$ translates into a unitary diagram containing the given contour $P$ and a spider arising from the universally quantified $x_1$ variable; this is $d_1$ in figure 33. The diagram $d_2$ constructs the set of elements to which $x_4$ is related and $d_3$ asserts that $x_2$ is not in that set. Thus, $d_2 \rightarrow d_3$ asserts $\neg R(x_4, x_2)$. The diagram $d_4$ trivially asserts $\neg (x_1 = x_3)$. Moving on to the second conjunct in the given symbolic sentence, $d_5$ is the translation of $\neg P(x_1)$ and $d_6 \rightarrow d_7$ translates $R(x_1, x_2)$.

There is no immediately obvious way of converting positive literals of the form $x = y$ into labels for a generalized constraint diagram; the example above illustrates that it is easy to convert sentences that do not involve such terms into generalized constraint diagrams. The technique needed for $x = y$ is a little more subtle and relies on the use of derived contours. In fact, for our translation, we need the set $DC$ to be countably infinite (after all, these contours are essentially acting as second order variables and it is typical to ensure that one never runs out of variables). The key observation here is that $x = y$ is equivalent to saying that there exists a set that contains exactly $x$ and exactly $y$. Thus, literals of the form $x = y$ are translated into diagrams that contain derived contours, shading and the spiders arising from $x$ and $y$. This is illustrated in figure 34, where we have assumed $x_1$ and $x_2$ are quantified as in the example above.

![Fig. 34. Translating $x_1 = x_2$.](image)

The translation process we have just described does not yield particularly elegant diagrams and one may often want to transform the resulting diagrams into more visually appealing, succinct diagrams. We note, though, that part of the reason for this inelegance is due to starting with a symbolic logic sentence in prenex normal form, with the quantifier free part being in disjunctive normal form. The diagrams we obtain under the specified translation are very similar in syntactic structure to these sentences. One would not typically construct sentences or diagrams with this structure to convey information.
Theorem 10.4  First order logic containing predicate symbols with arity at most two is at most as expressive as generalized constraint diagrams.

Proof [Sketch] Let $S$ be a first order logic sentence containing predicate symbols with arity at most two. First, convert $S$ into prenex normal form, say $Q_1x_1Q_2x_2\ldots Q_nx_nF$, where $F$ is in disjunctive normal form (the main point here is that all $\neg$’s are at the leaves and no $\Rightarrow$ symbols remain). Translate each literal in $F$ into a unitary diagram or collection of unitary diagrams as illustrated above and join the results together, respecting the connectives in $F$. It is then straightforward to show that the resulting diagram is equivalent to the original sentence. □

Corollary 10.1 The generalized constraint diagram language is undecidable.

It is unknown whether augmented constraint diagrams form an undecidable language although we conjecture this to be the case. The basis for this conjecture is that there are examples of augmented constraint diagrams that have only infinite models and, moreover, arbitrary quantifier alternations are permitted.

Theorem 10.4 essentially tells us that generalized constraint diagrams are capable of expressing as much of first order predicate logic as we can expect when we have only monadic and binary predicates. As an aside, an immediate consequence of this result is that, at least as far as first order statements are concerned, we have not reduced expressiveness by not including explicit negation and implication alongside our $\land$ and $\lor$ connectives. Some may find this surprising, since it is often thought that diagrammatic notations have inherent difficulties in expressing negated statements, for example. Generalized constraint diagrams are one of a family of notations for which explicit negation, by way of a $\neg$ operator, is syntactic sugar. Other examples include Shin’s Venn-II system [26] and spider diagrams [30].

Indeed, we argue that some negated statements are very naturally expressed using diagrammatic notations. For example, asserting the distinctness of elements requires explicit negation, $\neg(x = y)$, in symbolic languages as they are typically defined whereas distinctness tends to come for free in diagrammatic languages. Other examples of negated statements that can be asserted by generalized constraint diagrams include $A$ is not a subset of $B$, (which can also be asserted by the monadic languages Venn-II and spider diagrams), and $x$ is not related to $y$. Returning the focus to the expressiveness questions of this section, we have the following result for constraint diagrams.

Lemma 10.1 Let $d$ be a (non-generalized) constraint diagram. Then $d$ is semantically equivalent to some sentence in first order logic containing predicates with arity at most two.
Proof The semantics of $d$ are essentially a translation into such a sentence. ∎

Corollary 10.2 Constraint diagrams are at most as expressive as generalized constraint diagrams.

There is strong evidence to suggest that we have actually increased expressive power: we have illustrated examples of first order statements which, we conjecture, are not expressible by constraint diagrams (primarily those involving universally quantified, disjunctive statements). Moreover, the second order nature of generalized constraint diagrams further supports such a conjecture; one would need to establish that the notation is properly second-order in that there are statements not reducible to first order expressions although this is probably a trivial task.

To conclude this section, we note that a method to convert augmented constraint diagrams into generalized constraint diagrams can be extracted from our expressiveness work above. First, translate the augmented constraint diagram into a sentence containing predicate symbols with arity at most two. Next, convert that sentence in to prenex normal form, say $Q_1x_1 \ldots Q_nx_n G$ where $Q_1x_1 \ldots Q_nx_n$ is a block of quantifiers (and variables) and $G$ is quantifier free and in disjunctive normal form. Finally, follow the conversion method illustrated above to convert $Q_1x_1 \ldots Q_nx_n G$ into a generalized constraint diagram. We note that there are likely to be more elegant translations than that arising from this process.

11 Conclusion

In this paper, we have exposed a range of negative syntax interactions and counter-intuitive features of the constraint diagram language. The presence of such interactions is likely to impact the effective use of constraint diagrams in terms of reading, writing and reasoning about constraints. Furthermore, we also alluded to potential expressiveness limitations from a formal perspective. In total, this suggested that improvements to the notation are necessary if constraint diagrams are to be widely adopted as a modelling language. Consequently, we proceeded to generalize constraint diagrams, overcoming these counter-intuitive features and expressiveness limitations. We appreciate that drawing partial orders of diagrams to formulate a constraint may be considered a time consuming process. However, one can easily imagine that appropriate tool support could alleviate this, by automatically reproducing the previous diagram in the partial order which could then be edited by the user. Alternatively a tool could ‘record’ a user drawing a single diagram, which is then broken down into a partial order by the tool.
There is likely to be a balance between the optimal amount of syntax to be changed at a single step and the overall ‘length’ of the diagram in terms of ease of readability. A tool could aid the user identify differences between successive diagrams by moving the new pieces of syntax, akin to the types of graph movement discussed in [4], or using colour, or a dynamic view may be beneficial.

The way in which we have provided the formalization of the syntax allows us to make the concrete syntax domain specific if we so choose. For example, at the abstract level we have labelled ‘connective’ vertices by $\land$ or $\lor$ with the concrete visualization of them and their incident edges identifying the connective type by rendering them in different ways ($\leftarrow \downarrow$ versus $\rightarrow$). Instead, one could choose a common rendering of the edges at the concrete level and use labels (more closely matching the abstract level) to identify the connective type. Other choices could also be made for the way in which the edges are rendered visually, determined by the users of the notation and the context in which the notation is to be used. An obvious extension to our work is to allow the use of other connectives, such as $\Rightarrow$, or the operator $\neg$ but, as we have seen, this would not lead to an increase in (formal) expressiveness at a first order level.

There are various directions that we see this research taking. First, empirical studies are required to assess the relative strengths and weaknesses of constraint diagrams and their generalized version. The results of these studies will allow us to identify areas for further improvement to the notation. Moreover, empirical studies are required to compare these diagrammatic notations to textual languages such as the Object Constraint Language which was also designed with software specification as a target application area. These empirical studies should ultimately consider ease of reading, writing and reasoning as discussed in this paper. We do not expect any one notation to always outperform the others. Some users will have a preference for diagrammatic languages over symbolic languages and vice versa, for example.

This leads to the notion of heterogeneous notations which attempt to make a ‘universally good’ language by combining symbolic and diagrammatic aspects. The creation of such a notation is difficult. Given a diagrammatic notation and a symbolic notation (such as generalized constraint diagrams and OCL), a naïve starting point would be to identify what is ‘best’ about each of them and combine the best parts to create a heterogeneous language. However, this may result in a language which is more difficult to use than either of the notations with which one began because of how these pieces of syntax interact. Another naïve starting point would be to allow all of the diagrammatic and symbolic syntax to be present and combined in certain ways, but this may result in a notation that is too complex to be of use. Thus, substantial research is required in order to produce a heterogeneous notation that is better than its
diagrammatic and symbolic counterparts.

This paper presented a formalization of generalized constraint diagrams and established a lower bound on their expressiveness. An obvious next step is to define reasoning rules based on the notation. Such work was begun for constraint diagrams, for which a set of sound rules has been developed [8]; the discussions in [8] illustrate that defining reasoning rules for constraint diagrams is difficult. A (decidable) complete fragment of the language has been identified [28] where the reasoning rules are relatively simple to define. This fragment contains no universal spiders and, consequently, it essentially forms a fragment of the generalized constraint diagram language (of course, there would be some obligation to show that our formalization of the semantics is equivalent to that in [28]).

![Diagram](image)

Fig. 35. A reasoning step that changes unitary parts and the tree structure.

Reasoning rules for generalized constraint diagrams will fall into three main categories: those which make some change to a unitary part of a diagram; those corresponding to ‘logical’ manipulations of the tree structure such as distributivity; and rules that manipulate both the tree structure and the unitary parts. An example of reasoning that changes a unitary part can be seen in figure 20, where $d_7$ can be deduced from $d_6$ which in turn follows from $d_5$. An example that changes both the tree structure and the unitary parts can be seen in figure 35, where the disjunctive information in $d_2$ is split into two parts, $d_3$ and $d_4$. One can actually view this single reasoning step as being two reasoning steps, the first where the tree structure is changed, duplicating $d_2$ to give $d_1 \vdash d_2$ (this is like applying the symbolic rule $P \vdash P \lor P$) and then erasing parts of the existential spiders to give $d_1 \vdash d_4$. 

43
It has been observed that one of the advantages diagrams have over symbolic notations is related to reasoning: sometimes one must perform reasoning to make a deduction in the symbolic world whereas the deduction may come ‘for free’ in the diagrammatic case. Euler diagrams have been well-studied in terms of their free rides, such as with regard to subset relations as discussed in section 3. These free rides also, therefore, apply to the images of relations under domain restrictions in our generalized constraint diagrams. A further example of a free ride can be seen in figure 21, where the placement of an existential spider, e, inside Member and the relative positioning of Member and Film give the information that the element represented by e is not a film for free. This type of free ride is very similar to those exhibited by Euler diagrams.

Other examples of free rides also occur. In figure 36 the diagram asserts that \( \{x\}.f = A, A.g = \{y\} \) and \( \{y\}.h = \{x\} \) for some elements \( x \) and \( y \). We get, for free, that \( \{x\}.f.g = \{y\}, A.g.h = \{x\}, \{y\}.h.f = A, \{x\}.f.g.h = \{x\} \) and so forth.

The generalized version of constraint diagrams proposed in this paper can be further extended to incorporate more (proper) second order aspects. With our formalization, we already have the capability of talking about the existence of sets in arbitrary ways (through the introduction of a derived contour at an appropriate place in a generalized diagram). With a little further modification to the syntax (of unitary diagrams and the constraints imposed on the partial orders) and semantics, another type of derived contour could be used to make universally quantified statements involving sets. For example, the diagram \( d_1 \) in figure 37 contains a derived contour annotated by an asterisk, allowing us to make a statement about all sets of films. The diagram as a whole expresses that every set of films is associated with exactly one lead actor.

---

**Fig. 36.** Free rides involving arrows.

---

**Fig. 37.** Second order expressions.
References


[34] UK Ministry of Defence. The procurement of safety critical software in defence equipment, 1993. MOD Interim Standard 00-55.
