Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science 127 (2005) 53–69
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Investigating Reasoning with Constraint Diagrams

Andrew Fish[1,2] Jean Flower[1,3]

Visual Modelling Group
University of Brighton
Brighton BN2 4GJ, UK

Abstract

Constraint diagrams are a visual notation designed to express logical constraints. Augmenting the diagrams with a reading tree (effectively a partial ordering of quantifiers) ensures that each diagram has a unique semantic interpretation.
In this paper, we discuss examples of reasoning rules for augmented constraint diagrams which exhibit interesting properties or difficulties that can arise when developing rules for such a diagrammatic system. We do not present a complete set of rules, but investigate the generic problems arising, providing solutions. One problem corresponds to the nesting of quantifiers and another relates to the domain of universal quantification. These issues may be an important consideration in the definition of other logical reasoning systems which explicitly represent quantification diagrammatically.

Keywords: Diagrammatic reasoning, constraint diagrams, logical inference.

1 Introduction

The constraint diagram notation [6] has only recently been given full formal semantics [2,3]. It is intended that constraint diagrams, used to express logical constraints, will be used by software engineers alongside the Unified Modelling Language (UML). A specification of a software component, given in diagrammatic form, can be matched against a requirement specification, again

1 This research was partially supported by UK EPSRC grant GR/R63516
2 Email: Andrew.Fish@brighton.ac.uk
3 Email: J.A.Flower@brighton.ac.uk

doi:10.1016/j.entcs.2004.08.047

in diagrammatic form. This matching up of component against requirement can be done by providing a logical argument between pre and post conditions on the component behaviour. Once the constraint diagram system is given a set of reasoning rules, the argument can be conducted, presented and verified at the syntactic level.

The language of constraint diagrams is rich in its expressiveness, and we would like to be able to *reason* between constraint diagrams using diagram transformations called *reasoning rules*. A sensible approach to designing such rules is clear - users are thinking of deleting, adding and editing the syntax of the diagram. The rules must be valid (i.e. if $d1$ can be transformed into $d2$ using a sequence of reasoning rules then the semantics of $d1$ entail those of $d2$). In this paper we describe a selection of rules which delete, add or edit the diagram syntax. The rules have been chosen to illustrate interesting properties that will recur for other rules in this system and are of wider interest to developers of other diagrammatic systems.

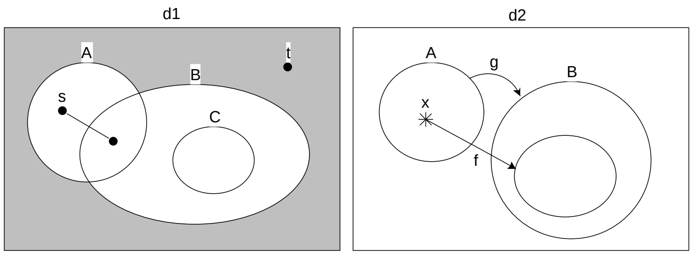

Fig. 1. Examples of constraint diagrams

Of course, reasoning could take place at the level of diagram semantics by writing proofs in first order predicate logic (FOPL). Some diagrammatic transformation rules correspond to simple FOPL transformations. However, some FOPL rules translate into complicated diagrammatic rules and some diagrammatic transformations correspond to non-trivial FOPL transformations. To maintain the advantage gained by providing a formal *diagrammatic* notation, we would like to reason between diagrams rather than between diagram semantics.

Work on reasoning rules for simple diagrammatic systems based on Euler diagrams has been undertaken [4,7,9]. A sound and complete reasoning system has been developed for spider diagrams [5], which form a subsystem of constraint diagrams (for an example of a spider diagram see $d1$ in figure 1). The expressiveness of spider diagrams is only equivalent to that of monadic first order predicate logic with equality [8], and so they are not very practical for use in software modelling, as opposed to the much more expressive constraint diagram system.

We will not present a complete set of reasoning rules for constraint diagrams, but we investigate the generic issues arising in the development of rules. These considerations may be useful in any attempt to define similar diagrammatic logical reasoning systems.

In sections 3 and 4, we describe rules which have strong preconditions and are designed to make small changes to the diagram syntax. Giving strong preconditions makes it easier to assess whether the rules are valid (assists reasoning *about* the system). This makes it easier to describe the changes that take place when the rule is applied, and simple rules are useful for automatic proof-writing algorithms. However, simple rules are applicable in fewer cases (hindering reasoning *with* the system).

In section 5 we show how to combine simple rules to make derived rules which are more widely applicable. These derived rules may have complex effects upon a diagram, and an algorithm is followed to determine the outcome. Such algorithms for rules could be implemented in a software tool. We imagine a user presented with a diagram would be able to select a syntactic element in a diagram and be offered applicable rules. While the effects of derived rules may be rather complex to describe, the burden of work falls upon the tool developers and not with the users.

2 Background work

Formal syntax and semantics of the constraint diagram system can be found in [2,3]. We illustrate the concepts by examples.

2.1 Constraint diagrams: syntax

In figure 1 there are two constraint diagrams. The first, $d1$, has three *given contours*, labelled A, B and C, and five *zones*, one of which is *shaded*. There are two *existential spiders*, s and t. Spider s has two *feet* and its *habitat* is the pair of zones inside A. Spider t has only one foot and its habitat is the shaded zone. The second diagram, $d2$, includes a *universal spider*, x, with habitat inside A and two *arrows* labelled f and g. The arrow labelled f is *sourced* at x and has a *derived contour* as its *target*. Derived contours have no labels.

Constraint diagrams as described above can be ambiguous [3]. Augmenting the diagrams with a reading tree (essentially a partial order on the spiders in the diagrams) ensures a unique semantic interpretation: a *reading* which is a sentence in FOPL. In this paper, when we refer to a constraint diagram, we really refer to an *augmented constraint diagram*, which is a constraint diagram accompanied by a reading tree, as defined in [3].

2.2 Constraint diagrams: semantics

Informally, we describe the semantics of these pieces of syntax. Given contours represent sets and derived contours represent the image of a relation. Topological properties, such as disjointness and containment of contours are respected by the corresponding sets. Existential spiders represent existential quantification, universal spiders represent universal quantification and arrows represent relations. Shading in a zone places an upper bound on the cardinality of the corresponding set. Distinct spiders represent distinct elements.

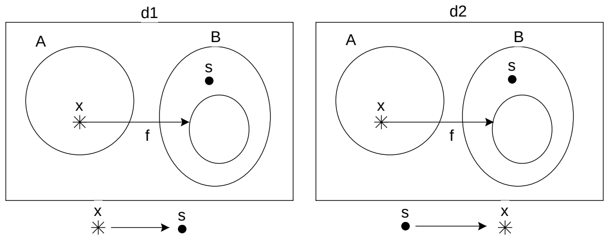

Fig. 2. Constraint diagrams

Example 2.1 In figure 2, the diagrams $d1$ and $d2$ have different reading trees. The readings for $d1$ and $d2$ are

$$A \cap B = \emptyset \wedge \forall x \in A(x.f \subseteq B \wedge \exists s \in B - x.f)$$

$$\text{and } A \cap B = \emptyset \wedge \exists s \in B(\forall x \in A(x.f \subseteq B - \{s\})).$$

In these expressions, the notation $x.f$ is used to denote the relational image of x under the relation f.

The first part of these readings: $A \cap B = \emptyset$ is known in general as *the plane tiling condition* [3,5]. It expresses the information we can deduce from the relative positioning of the given contours. In this case A and B are disjoint.

All readings start with the plane tiling condition, but in this paper we have chosen to omit this part of the reading, for brevity.[4] The derived contours are not included in the plane tiling condition, and they are interpreted later in the reading (e.g. $x.f \subseteq B$). A derived contour may represent a collection of sets: different sets $x.f$ for different values of x.

[4] Readers may notice that some reading trees appear to be disconnected (see figure 3, for example). This is because each tree includes a root node, corresponding to the plane tiling condition, which we omit from the figures. Furthermore, in this paper, any reading tree which consists of only the root node and one other node is omitted.

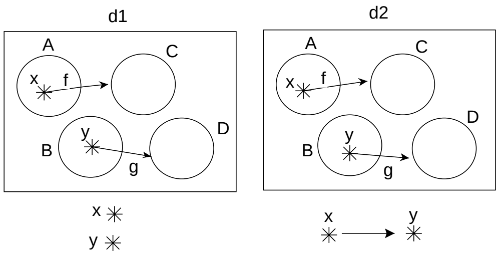

Fig. 3. The effect of ordering nodes x and y

Example 2.2 In figure 3, the readings of $d1$ and $d2$ are

$$\forall x \in A(x.f = C) \wedge \forall y \in B(y.g = D)$$
$$\text{and } \forall x \in A(x.f = C \wedge \forall y \in B(y.g = D)).$$

In the reading tree of $d1$ the nodes x and y are unordered and so the quantifiers x and y are not nested in the reading. In $d2$, x is ordered before y and so the quantifiers are nested: y is in the scope of x. Therefore, the reading of $d2$ is true whenever the set corresponding to A is empty, whatever the relationship between y, g and D, but this is not the case for the reading of $d1$.

Diagram readings are FOPL sentences which may include phrases like $\forall y \in A(...)$, which is a shorthand version of $\forall y(y \in A \Rightarrow ...)$. These sentences are the kinds of assertions needed when modelling software systems in order to make statements about all objects of a certain type. When designing reasoning rules, we have to be constantly aware of the case where universal quantification can take place over an empty *domain* (the set A above).

2.3 Compound diagrams

We can build *compound* diagrams from unitary diagrams using the logical connectives *and*, *or* and *not*.

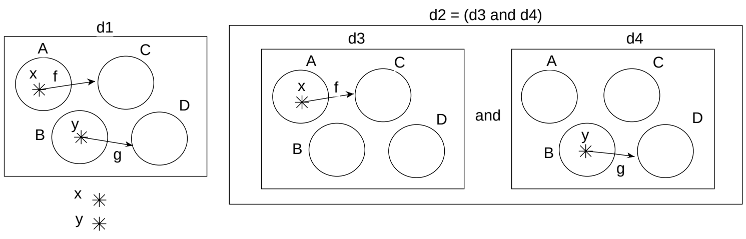

Fig. 4. Compound constraint diagrams

Example 2.3 In figure 4, $d1$ is unitary, and is semantically equivalent to the compound diagram $d2 = d3 \wedge d4$. Both have reading

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

2.4 Dependence of spiders and ordering in the reading tree

Dependence between spiders in a diagram is a syntactic condition which encapsulates the need of the corresponding quantifiers to reference each other in the semantic interpretation. An *ordering* of nodes in a reading tree for a diagram places one of the corresponding quantifiers in the scope of the other. Say that a reading tree is *valid* for a diagram if it provides ordering between the nodes of dependent spiders. It may or may not provide ordering between independent spiders.

For example, in figure 2, spiders x and s are dependent in the diagrams, and so their nodes have to be ordered in the reading trees (allowing us to say $s \in B - x.f$, for example). However, in figure 3, the spiders x and y are independent and so they don't have to be ordered in the tree.

3 Diagrammatic rules

Rules discussed in this section are purely diagrammatic, transforming a unitary diagram into another unitary diagram. These simple rules have strong preconditions and are designed to make small changes to the diagram syntax. We have, in some cases, written preconditions that are stronger than they need to be, in order to be able to describe the rule simply in a limited amount of space.

3.1 Erase shading

This rule can reduce dependence between spiders. There is no precondition on the rule; the existing reading tree will be valid after an application of the rule.

Example 3.1 In figure 5 the diagrams $d1$ and $d2$ have readings

$$\forall x \in A(x.f \subseteq \overline{A} \cap \overline{B} \wedge \forall y \in B(y.g \subseteq \overline{A} \cap \overline{B} \wedge x.f \cap y.g = \emptyset)),$$

$$\text{and } \forall x \in A(x.f \subseteq \overline{A} \cap \overline{B} \wedge \forall y \in B(y.g \subseteq \overline{A} \cap \overline{B})).$$

In $d1$, the spiders x and y are dependent because of the shading. Both quantifiers must be in scope in order to assert that the shaded area is empty ($x.f \cap y.g = \emptyset$). In $d2$, the spiders are independent.

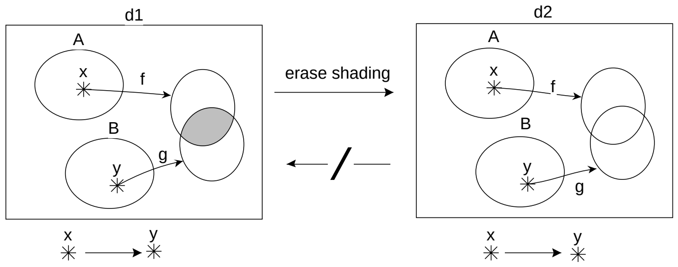

Fig. 5. Erasing shading

3.2 Add existential spider feet

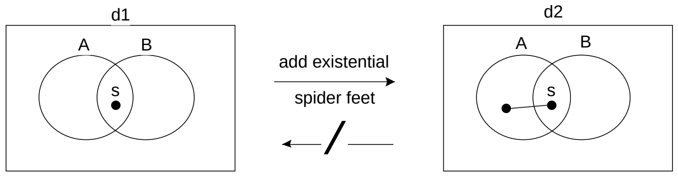

Fig. 6. Adding existential spider feet

Example 3.2 Figure 6 shows the transformation of $d1$ with reading $\exists s \in A \cap B$ into $d2$ with reading $\exists s \in A$. This rule has enlarged the domain of s.

Enlarging the domain of a spider can introduce new dependencies. This provides a precondition on the rule: the reading tree for the premise diagram has to be valid for the conclusion diagram.

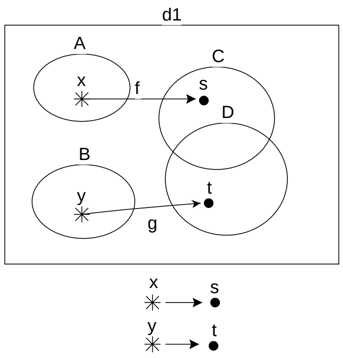

Fig. 7. Existential spiders that cannot have certain feet added

Example 3.3 Figure 7 shows a case where neither of the existential spiders s nor t can be extended into the habitat of the other. The reading tree has s

and t unordered and so the scopes of s and t are disjoint in the reading

$$\forall x \in A(\exists s \in C-D(s = x.f)) \wedge \forall y \in B(\exists t \in D-C(t = y.g)).$$

If one tried to add a foot to s in the zone occupied by t, then there would be no way to say that $s \neq t$ because in this reading there is *insufficient nesting of quantifiers.*

In both of the preceding examples, we considered the addition of single feet. Some spiders can have multiple feet added, even though those feet cannot be added individually.

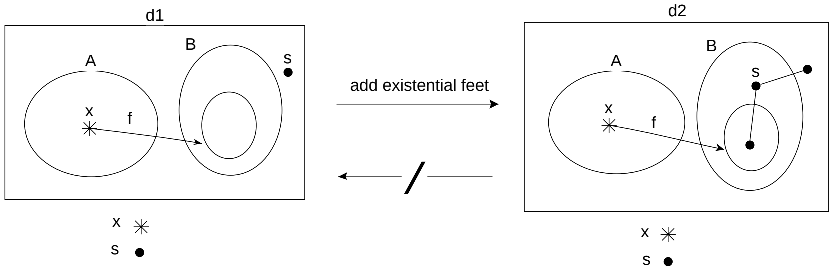

Fig. 8. Adding multiple feet to an existential spider

Example 3.4 In figure 8, the readings of $d1$ and $d2$ are

$$\forall x \in A(x.f \subseteq B) \wedge \exists s \in \overline{A} \cap \overline{B},$$

$$\text{and } \forall x \in A(x.f \subseteq B) \wedge \exists s \in \overline{A}.$$

These feet could not have been added individually because x is not in the scope of s (nor vice versa) and we can not say that $\exists s \in \overline{A} \cap \overline{x.f}$, for example.

3.3 *Delete existential spider*

An existential spider s can be deleted if none of the following conditions hold:

- s is the source or target of any arrow,
- the habitat of s includes a shaded zone,
- there is a universal spider x whose domain includes a foot of the spider s and the node s is ordered before the node x in the tree.

Example 3.5 In figure 9, the readings of $d1$ and $d2$ are

(1) $$\forall x \in A(\exists s \in A(s \neq x \wedge \exists t \in \overline{A}(x.f = t))),$$

(2) $$\text{and } \forall x \in A(\exists t \in \overline{A}(x.f = t)).$$

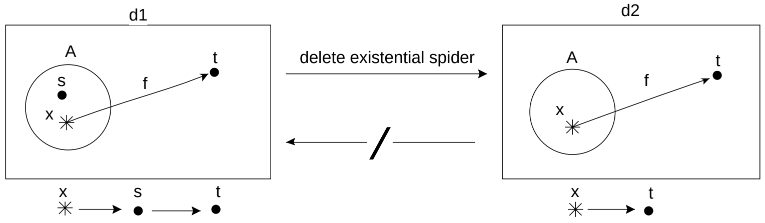

Fig. 9. Deleting an existential spider

In $d1$, s is ordered after x in the tree, so s can be deleted from $d1$ to give $d2$. If the reading tree of $d1$ had been $s \to x \to t$ then the reading would have been

$$\exists s \in A(\forall x \in A((s = x) \vee (\exists t \in \overline{A}(x.f = t)))) \tag{3}$$

and we would not have been able to delete s from the diagram.

We cannot deduce expression 2 from expression 3 because we would be inferring the stronger property $\exists t \in \overline{A}(x.f = t)$ from the weaker property $(s = x) \vee (\exists t \in \overline{A}(x.f = t))$. This *property strengthening* prevents us from removing the spider s.

This is the first rule to affect the reading tree. The resulting tree is obtained by first adding edge $a \to b$ whenever there are edges $a \to s$ and $s \to b$, and then deleting s from the tree.

3.4 Delete arrow

We can always delete an arrow which does not target a derived contour.

3.4.1 Derived contour targeted by a single arrow

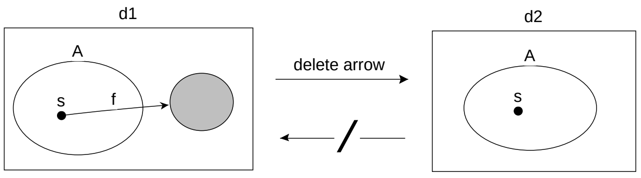

Fig. 10. Deleting an arrow can delete a derived contour

If an arrow f targets a derived contour c, and no other arrow targets c then c will be deleted when f is deleted.[5]

[5] After the deletion of a contour any partial shading (not occupying a complete zone) is erased and if there is a spider with more than one foot in a zone, those feet will be merged.

Example 3.6 In figure 10, $d1$ and $d2$ have readings

$$\exists s \in A(s.f \subseteq \overline{A} \wedge s.f = \emptyset) \quad \text{and} \quad \exists s \in A.$$

The three preconditions on deletion of such an arrow and contour are:

- there are no arrows emanating from the target contour,
- the derived contour is not needed to describe the domain of any universal spider,
- the reading tree for the premise diagram is valid for the conclusion diagram.

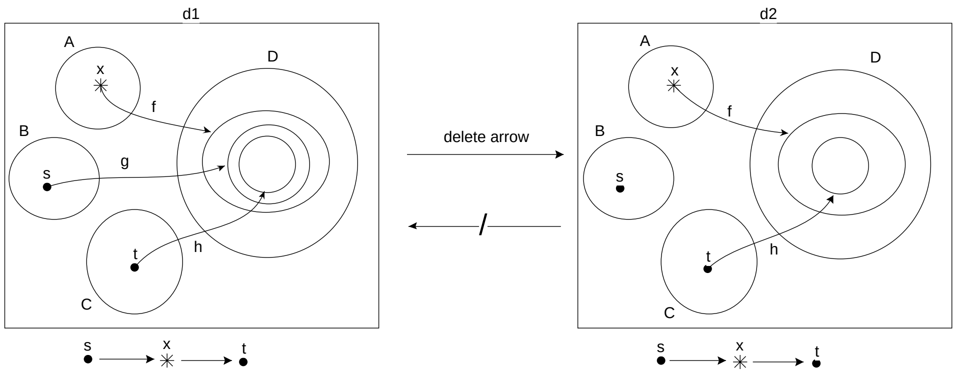

Fig. 11. Deleting an arrow can introduce dependence

Example 3.7 In figure 11 deleting g will delete the derived contour $s.g$, introducing a new dependence between x and t. The readings of $d1$ and $d2$ are

$$\exists s \in B(s.g \subseteq D \wedge \forall x \in A(s.g \subseteq x.f \subseteq D \wedge \exists t \in C(t.h \subseteq s.g))),$$
$$\text{and } \exists s \in B(\forall x \in A(x.f \subseteq D \wedge \exists t \in C(t.h \subseteq x.f))).$$

Because the reading tree is still valid with the new dependence between x and t, we can delete g. If the reading tree had been $x \leftarrow s \rightarrow t$ (branched at s with x and t unordered) then the reading would have been

$$\exists s \in B(s.g \subseteq D \wedge \forall x \in A(s.g \subseteq x.f \subseteq D) \wedge \exists t \in C(t.h \subseteq s.g)).$$

and g could not have been deleted from the diagram using this simple rule because there is *insufficient nesting of the quantifiers* in the reading (x and t are not nested).

3.4.2 Derived contour targeted by multiple arrows

Suppose that a derived contour is the target of more than one arrow. The only precondition for *delete arrow* in this case is that we do not enlarge the domain of any universal spider.

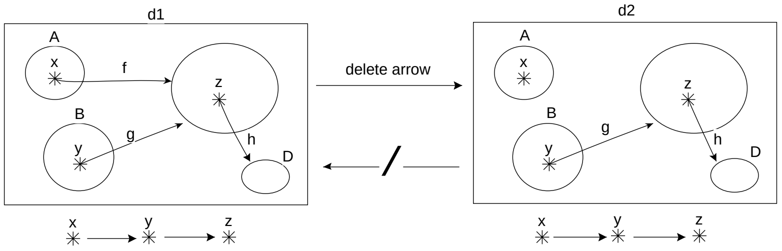

Fig. 12. Deleting an arrow can delay the interpretation of a derived contour

Example 3.8 In figure 12, the readings of $d1$ and $d2$ are

$$\forall x \in A(x.f \subseteq \overline{A} \cap \overline{B} \cap \overline{D} \wedge \forall y \in B(y.g = x.f \wedge \forall z \in x.f(z.h = D)))$$

and $\forall x \in A(\forall y \in B(y.g \subseteq \overline{A} \cap \overline{B} \cap \overline{D} \wedge \forall z \in y.g(z.h = D)))$.

In $d1$, f targets a derived contour c, and f is used to *interpret* the contour (in the expression $x.f \subseteq \overline{A} \cap \overline{B} \cap \overline{D}$). The contour c is interpreted using f rather than g because x was ordered before y in the reading tree. In $d2$, after deleting f, the derived contour c is interpreted later in the sentence ($y.g \subseteq \overline{A} \cap \overline{B} \cap \overline{D}$) and $y.g$ is used in place of $x.f$ to refer to c in other assertions (such as $\forall z \in y.g(z.h = D)$). Say that we *delayed the interpretation* of c.

This delay of interpretation was possible because the reading tree ordered y before z. If the tree had been $x \to z \to y$ then the readings would have been:

$$\forall x \in A(x.f \subseteq \overline{A} \cap \overline{B} \cap \overline{D} \wedge \forall z \in x.f(z.h = D \wedge \forall y \in B(y.g = x.f))),$$

$$\forall x \in A(\forall z \in \overline{A} \cap \overline{B} \cap \overline{D}(z.h = D \wedge \forall y \in B(y.g \subseteq \overline{A} \cap \overline{B} \cap \overline{D} \wedge z \in y.g)))$$

and the removal of f would have *enlarged the domain* of z to $\overline{A} \cap \overline{B} \cap \overline{D}$.

3.5 Delete universal spider

A universal spider x cannot be deleted if there are any arrows sourced at x or targeted on x. A second precondition asserts that no spider can be ordered after x in the reading tree. This is because in the absence of any contextual information, the universal quantification could have an *empty domain*. If a universal spider x has another spider s ordered after x in the reading tree, then the reading takes the form $...\forall x \in A(...\exists s \in B(...))$. If the set A might be empty, we can not replace this by $...\exists s \in B(...)$.

3.6 Transpose existential node labels

Suppose that in a reading tree there is exactly one edge emanating from existential spider node s, which targets existential spider node t. Then the labels s and t can be swapped.

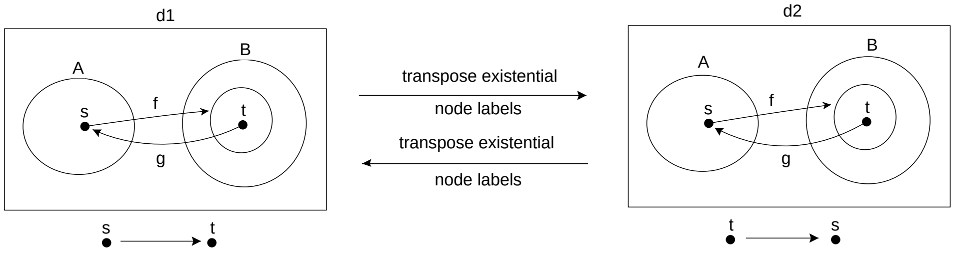

Fig. 13. Transposing existential node labels

Example 3.9 The readings of $d1$ and $d2$ in figure 13 are equivalent:

$$\exists s \in A(s.f \subseteq B \wedge \exists t \in s.f(t.g = s)),$$

$$\text{and } \exists t \in B(\exists s \in A(s = t.g \wedge t \in s.f \subseteq B)).$$

3.7 Move branch

Suppose that in a reading tree there are edges from spider p to spider q and from p to an existential spider s. Then the *move branch* rule deletes the edge from p to q and adds an edge from s to q.

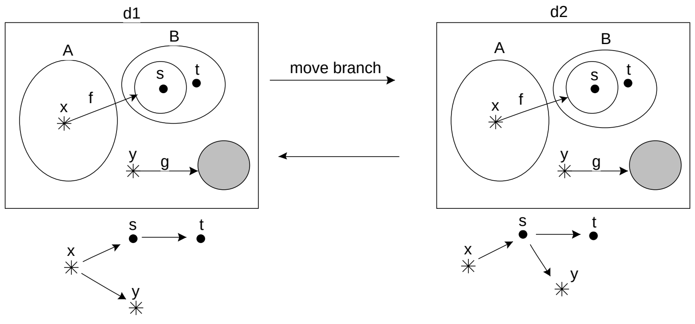

Fig. 14. Moving a tree branch

Example 3.10 The readings of $d1$ and $d2$ in figure 14 differ only by the placing of brackets

$$\forall x \in A(x.f \subseteq B \wedge \exists s \in x.f(\exists t \in B - x.f) \wedge \forall y \in \overline{A} \cap \overline{B}(y.g = \emptyset))$$
$$\forall x \in A(x.f \subseteq B \wedge \exists s \in x.f(\exists t \in B - x.f \wedge \forall y \in \overline{A} \cap \overline{B}(y.g = \emptyset)))$$

4 Rules which are diagrammatic and structural

The rules in this section change the syntax within unitary parts of a diagram, as well as the propositional-logic structure of the diagram.

4.1 Excluded middle

The excluded middle rule allows us to explicitly express the fact that a set either has no additional elements or some additional elements.

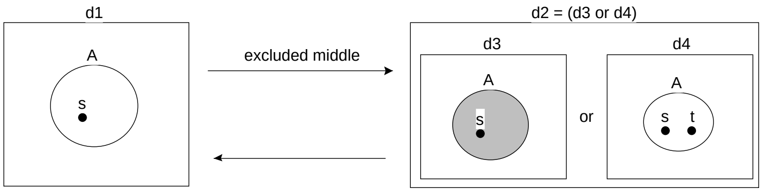

Fig. 15. An example of the excluded middle rule

Example 4.1 In figure 15, $d1$ and $d2 = d3 \vee d4$ are semantically equivalent. They have readings $\exists s \in A$ and $\exists s \in A(A - s = \emptyset) \vee \exists s \in A(\exists t \in A(t \neq s))$.

The precondition for the *excluded middle* rule is that the zone it is applied to can be described without reference to a universal spider.

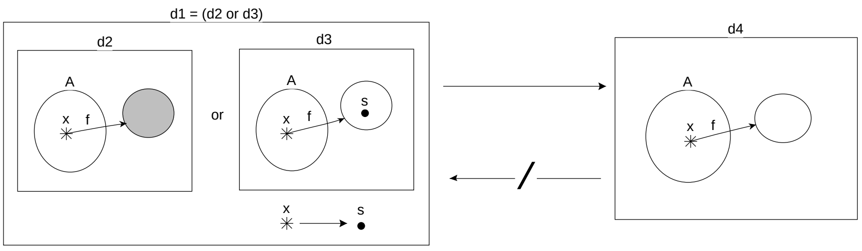

Fig. 16. Excluded middle does not apply

Example 4.2 In figure 16, the readings of $d1$ and $d4$ are

$$\forall x \in A(x.f \subseteq \overline{A} \wedge x.f = \emptyset) \vee \forall x \in A(x.f \subseteq \overline{A} \wedge \exists s \in x.f),$$

$$\text{and } \forall x \in A(x.f \subseteq \overline{A}).$$

The diagram $d4$ can be obtained from $d1$ by erasing shading from $d2$, deleting the spider s in $d3$ and using idempotency of $\vee$.

Diagrams $d2$ and $d3$ only differ from $d4$ by the addition of extra shading and an extra existential spider in $x.f$. But the disjunction $d1 = d2 \vee d3$ cannot be deduced from $d4$, because, while it *is* true that, for each x, $(x.f = \emptyset)$ or $(\exists s \in x.f)$ holds, it is not necessarily the case that $(x.f = \emptyset)$ for every x, or $(\exists s \in x.f)$ for every x.

4.2 Split existential spider

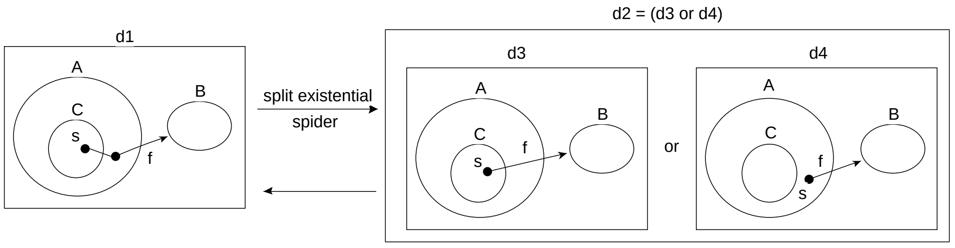

Fig. 17. Splitting an existential spider

Example 4.3 The readings of $d1$ and $d2$ in figure 17 are $\exists s \in A(s.f = B)$ and $\exists s \in C(s.f = B) \vee \exists s \in A - C(s.f = B)$. These statements are equivalent (because $C \subseteq A$ from the plane tiling condition).

An existential spider s can be *split* (see figure 17) provided that whenever s is ordered after a universal spider x in the reading tree, the spiders s and x are independent in the diagram.

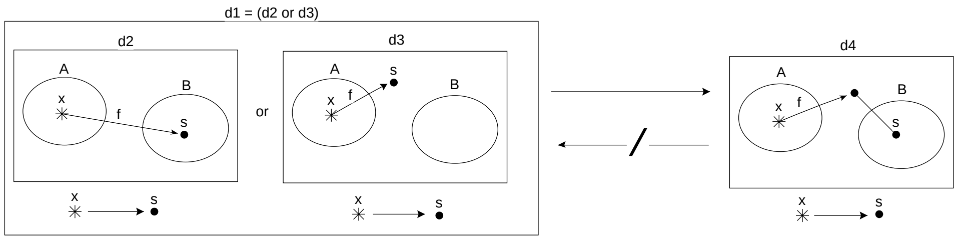

Fig. 18. Split spider is not always applicable

Example 4.4 The readings of $d1$ and $d4$ in figure 18 are

$$\forall x \in A(\exists s \in B(x.f = s)) \vee \forall x \in A(\exists s \in \overline{A} \cap \overline{B}(x.f = s)),$$

and $\forall x \in A(\exists s \in \overline{A}(x.f = s))$.

These two sentences are not equivalent, but $d1$ entails $d4$. The spider s is ordered after universal spider x in the tree but s and x are dependent, preventing us from reasoning from $d4$ to $d1$.[6] We can reason from $d1$ to $d4$ by applying add *existential spider foot* to s in both $d2$ and $d3$, followed by an application of *idempotency of* $\vee$.

5 Derived rules for tool-building

The rules in the previous section were designed to make only small changes to the diagram syntax and have stringent preconditions. However, we anticipate that users of the constraint diagram notation will prefer to use rules which can be applied more widely. If, for example, a user wishes to delete a given contour from a diagram, it is not very useful to refuse that request simply because of some failed precondition. A better response would be to automate the application of other rules like *delete existential spider* or *delete arrow* in order to prepare the diagram so that it complies with the relevant preconditions, where possible.

It is a non-trivial task to determine which syntactic elements need to be edited or deleted, and in which order, to prepare a diagram for a rule application. We do *not* expect this task to be undertaken by the users of the notation. It is intended to be implemented as part of a software tool for reasoning with constraint diagrams. The burden of work is with the tool-builders, and the users can simply apply rules using the tool which invokes algorithms behind the scenes.

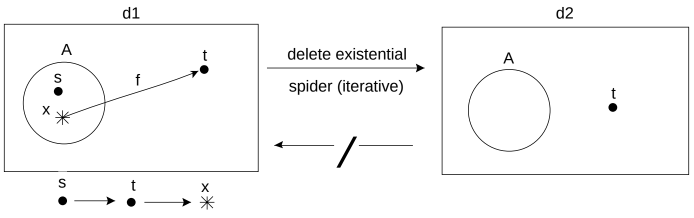

Fig. 19. Iterative *delete spider*

[6] In the spider diagram system, a useful step in proof strategies and rule development was the transformation of diagrams into α-*diagrams* in which all spiders have only one foot. The precondition on *split existential spider* prevents us from guaranteeing that every diagram can be transformed into an α-*diagram*.

Example 5.1 In figure 19, we revisit example 3.5. Spider s cannot be deleted from $d1$ using the *delete existential spider* rule defined in section 3.3 because s is ordered before x in the tree and s has a foot in the domain of x. To enable deletion of s, we have to delete x. In turn, in order to delete x, we have to delete f.

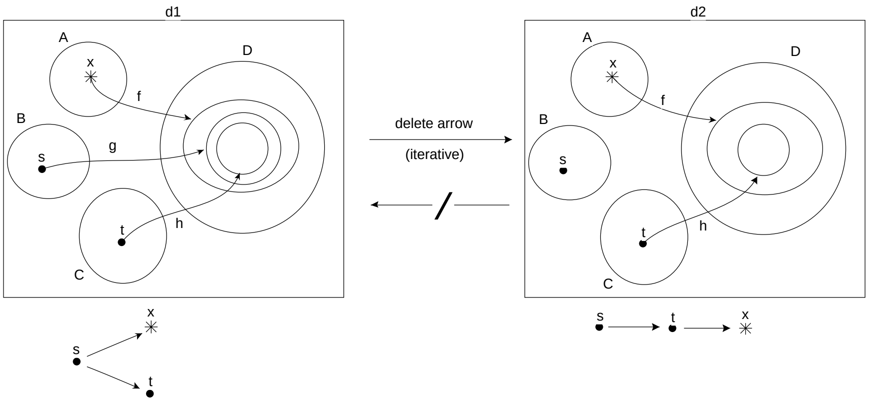

Fig. 20. Iterative *delete arrow*

Example 5.2 In figure 20 we revisit example 3.7. The preconditions of *delete arrow* as defined in section 3.4 prevent us from deleting g, because it would introduce new dependence between spiders x and t in the diagram, but these are unordered in the tree. However, we can apply the *move branch* rule which orders x and t in the tree and then we can delete g.

6 Conclusion and further work

This work is progress towards the development of a sound and complete diagrammatic reasoning system which is sufficiently expressive to be useful for software modelling. We highlight issues and subtleties which recur as valid rules are developed.

The following semantic issues will recur in the development of reasoning rules for other logical systems which explicitly express quantification diagrammatically:

- insufficient nesting of quantifiers,
- enlarged domain of quantification,
- property strengthening,
- empty domains.

The rules we have described have been given syntactic preconditions designed to resolve these semantic issues. The syntactic conditions may change from one rule to another, even if they are addressing the same semantic issue.

There are conflicting pressures concerning the simplicity of the rules. One group of users, those who are reasoning *about* the system (e.g. proving soundness, or generating automatic proof-writers) would like to use rules which make minimal changes to diagram syntax. However, such rules need to have stringent preconditions and can rarely be applied to a diagram.

A second group of users reason *with* diagrams to formulate software specifications. A practical software tool would allow such users to apply rules easily. The software should be responsible for offering rules only when preconditions apply, and be able to provide conclusion diagrams which conform to post-conditions. For these users, the most useful kinds of rules have weak preconditions and complex postconditions. To balance these conflicting requirements, we developed two classes of rules: simple and iterative.

Future plans include defining reasoning rules in terms of graph transformations, which may enable us to make use of existing results and tooling environments (see, for example, [1]) to conduct reasoning with constraint diagrams.

References

[1] *AGG: The Attributed Graph Grammar System.* Chief researcher: Gabi Taentzer. Web page `http://tfs.cs.tu-berlin.de/agg/`, accessed 12th October 2004.

[2] A. Fish, J. Flower and J. Howse. *A reading algorithm for constraint diagrams.* Symp. on human-centric computing languages and environments, pages 161–168, IEEE, 2003.

[3] A. Fish, J. Flower and J. Howse. *The semantics of augmented constraint diagrams.* Submitted to JVLC, 2004.

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

[5] J. Howse, F. Molina and J. Taylor. *SD2: A sound and complete diagrammatic reasoning system.* Symp. on Visual Languages, pages 127–136, IEEE, 2000.

[6] S. Kent. *Constraint diagrams: Visualising invariants in object oriented models.* OOPSLA97, SIGPLAN Notices, vol. 32, no. 10, pages 327–341, ACM, 1997.

[7] S.-J. Shin. *The Logical Status of Diagrams.* Cambridge University Press, 1994.

[8] G. Stapleton, J. Howse, J. Taylor and S. Thompson. *What can spider diagrams say?* International conference on the theory and application of diagrams, LNAI 2980, pages 122–127, Springer-Verlag, 2004.

[9] N. Swoboda. *Implementing Euler/Venn reasoning systems.* In *Diagrammatic Representation and Reasoning*, M. Anderson, B. Meyer and P. Oliver, eds, pages 371–386, Springer-Verlag, 2001.