Generalized constraint diagrams: the classical decision problem in a diagrammatic reasoning system

James Burton

A thesis submitted in partial fulfilment of the requirements of the University of Brighton for the degree of Doctor of Philosophy

May 2011

The University of Brighton

Abstract

Constraint diagrams are part of the family of visual logics based on Euler diagrams. They have been studied since the 1990s, when they were first proposed by Kent as a means of describing formal constraints within software models. Since that time, constraint diagrams have evolved in a number of ways; a crucial refinement came with the recognition of the need to impose a reading order on the quantifiers represented by diagrammatic syntax. This resulted first in augmented constraint diagrams and, most recently, generalized constraint diagrams (GCDs), which are composed of one or more unitary diagrams in a connected graph. The design of GCDs includes several syntactic features that bring increased expressivity but which also make their metatheory more complex than is the case with preceding constraint diagram notations. In particular, GCDs are given a second order semantics.

In this thesis we identify a decidable fragment of GCDs and provide the first set of sound inference rules for the system. We define a particular class of the unitary diagrams drawn from this fragment, which we call γ-diagrams. We describe a decision procedure for the satisfiability of unitary γ-diagrams, before developing a means of applying the decision procedure to all unitary diagrams of the fragment, achieved by using the class of γ-diagrams as a reduction class. Next, we develop a decision procedure for the non-unitary diagrams of the fragment. This procedure makes use of several normal forms which enable us to judge the satisfiability of a (non-unitary) generalized diagram by examining the unitary diagrams it contains. We discuss the ways in which our work is of benefit to users of GCDs and those engaged in making software tools based on them. Finally, we identify the ways in which our results provide the foundations for further theoretical study of the system.

Contents

Contents **iii**

1 Introduction **1**

1.1 Related work . . . 2

1.1.1 Constraint diagrams . . . 8

1.1.2 The decision problem in symbolic and visual logic . . . 15

1.1.3 Decidability in monadic languages . . . 18

1.2 Thesis outline . . . 26

2 Syntax and semantics **29**

2.1 Introduction . . . 29

2.1.1 Restrictions imposed on generalized constraint diagrams in our system . . . 33

2.2 Syntax . . . 33

2.2.1 Generalized unitary constraint diagrams . . . 34

2.2.2 Generalized constraint diagrams . . . 38

2.3 Semantics . . . 44

3 The unitary decision procedure **54**

3.1 Satisfiability for unitary diagrams . . . 54

3.1.1 Inconsistency . . . 55

3.1.2 Standard interpretations . . . 64

3.1.3 A decision procedure for the satisfiability of unitary γ-diagrams . . . 70

3.2 Transformations . . . 70

3.2.1 Transformations that remove syntax . . . 72

3.2.2 Transformations that add items of syntax 78
3.2.3 Completeness of the unitary transformations 83
3.3 Extending to the unitary fragment 85
3.3.1 Inference rules 87
3.3.2 Validity of the inference rules 90
3.3.3 Transforming α-diagrams to sets of γ-diagrams 94

4 The decision procedure for the existential fragment 102
4.1 Transformations 106
4.1.1 Atomic transformations 107
4.1.2 Derived transformations 115
4.2 $\wedge$-linear normal form 125
4.3 Pushing syntax 131
4.3.1 Preparations for pushing syntax 134
4.3.2 Inference rules which push syntax 137
4.3.3 Validity of the inference rules which push syntax 150
4.3.4 Pushed syntax normal form 162
4.4 Disjunctive normal form 168
4.4.1 Removing non-linear $\vee$-labelled nodes 171
4.4.2 The trivial prefix 174
4.4.3 Validity of the inference rules required by disjunctive normal form 177
4.4.4 Obtaining disjunctive normal form 182
4.5 Reducing linear generalized diagrams 187
4.5.1 Removing syntax from a linear diagram 190
4.5.2 Validity of the inference rules required to reduce linear diagrams 194
4.5.3 A decision procedure for linear generalized diagrams 199
4.6 The decision procedure 201

5 Conclusions and further work 205
5.1 Results and contribution of this thesis 205
5.2 Further work 210
5.2.1 The relationship between the existential fragment and the full system 210

5.2.2 Implementing the decision procedure 212
5.2.3 Usability 214
5.2.4 Completeness 215

Glossary **232**

List of Figures

1.1 Two Euler diagrams. 3
1.2 A proof using Euler diagrams. 3
1.3 A Venn diagram. 4
1.4 A Peirce diagram. 5
1.5 Venn-I. 5
1.6 Venn-II. 6
1.7 A spider diagram. 7
1.8 A constraint diagram. 9
1.9 An ambiguous constraint diagram. 10
1.10 An augmented constraint diagram. 10
1.11 A generalized constraint diagram. 11
1.12 Disjunction in generalized constraint diagrams. 12
1.13 Disjunction and conjunction in generalized constraint diagrams. . 12
1.14 Free rides 13
1.15 An undecidable generalized constraint diagram. 14
1.16 A diagram from the existential fragment. 15
1.17 Consistency in constraint diagrams and the generalized case . . . 17
1.18 Zones and spiders in spider diagrams. 20
1.19 The abstract syntax of spider diagrams. 22
1.20 A compound spider diagram. 23
1.21 Interpreting a spider diagram. 24

2.1 Introducing constraint diagrams. 30
2.2 The syntax of constraint diagrams. 30
2.3 A compound constraint diagram. 32
2.4 A generalized constraint diagram. 32

2.5 Reading order in generalized constraint diagrams. 33
2.6 Sources and targets of arrows. 34
2.7 Illustrating ties between spiders. 35
2.8 Zones containing a contour and the spiders inhabiting those zones. 37
2.9 Generalized diagrams and scope. 38
2.10 The structure of a generalized diagram. 39
2.11 Visualising the structure of a generalized diagram. 40
2.12 Diagrams and sub-diagrams. 42
2.13 Linear and non-linear connectives. 43
2.14 The ancestors and descendants of a node. 44
2.15 Interpretations of a unitary diagram. 45
2.16 Illustrating GCD semantics. 49
2.17 Interpreting a GCD. 52

3.1 A γ-diagram. 55
3.2 Three examples of inconsistency arising from arrows. 56
3.3 Source and target inconsistency arising together. 57
3.4 Interaction of multiple arrows. 58
3.5 Target inconsistency. 59
3.6 The need for γ-diagrams. 60
3.7 A spider in the target of one of the arrows of X. 62
3.8 The semantic implication of Figure 3.7. 63
3.9 Constructing a model with too few source spiders. 64
3.10 Restricting the mapping for arrow labels. 66
3.11 Purely syntactic transformations. 71
3.12 An invalid transformation. 71
3.13 Transforming a diagram by removing an arrow. 72
3.14 Transforming a diagram by removing shading from a zone. 73
3.15 Transforming a diagram by removing a spider. 73
3.16 Transforming a diagram by removing a tie. 74
3.17 Transforming a diagram by removing a zone. 76
3.18 Transforming a diagram by removing a contour. 77
3.19 Transforming a diagram by adding an arrow. 78
3.20 Transforming a diagram by adding an spider to a zone. 79

3.21 Transforming a diagram by adding a tie between two spiders. . . . 80
3.22 Transforming a diagram by adding shading to a zone. 81
3.23 Transforming a diagram by adding a zone. 81
3.24 Transforming a diagram by adding a contour. 82
3.25 Removing spiders to obtain singleton-τ form. 85
3.26 β-diagrams and contour-source form. 87
3.27 An application of *remove equal spider*. 87
3.28 An application of *add contour over spider*. 89
3.29 An application of *replace arrow*. 90
3.30 Equivalence class representatives of τ. 95
3.31 Obtaining contour-source form. 98
3.32 An α-diagram and its γ components. 99

4.1 The normal forms defined in this chapter. 102
4.2 A running example: before manipulation into normal form. 103
4.3 A running example: after manipulation into normal form. 104
4.4 The trivial diagram. 105
4.5 Transforming a generalized diagram by removing a sub-diagram. . 106
4.6 An invalid compound transformation. 106
4.7 The sub-diagram induced by a particular node. 107
4.8 Removing a sub-diagram. 107
4.9 Transforming a diagram by removing the root node. 109
4.10 Transforming a diagram by removing a non-root node. 109
4.11 One of the immediate neighbours must be linear when removing a node. 110
4.12 Attaching a diagram to a leaf node. 112
4.13 Attaching a diagram to a connective-labelled node. 113
4.14 Relabelling a node. 113
4.15 Attaching a set of diagrams to a connective-labelled node. 116
4.16 Attaching a set of diagrams to a leaf node. 117
4.17 Prior to attaching a set of diagrams to every leaf node. 117
4.18 The result of attaching a set of diagrams to every leaf node. . . . 118
4.19 Replacing a sub-diagram. 119
4.20 Preparing to replace a sub-diagram by a set of diagrams. 119

4.21 The result of replacing a sub-diagram. 120
4.22 Replacing an inner node. 120
4.23 Replacing an inner node: the first intermediate steps. 121
4.24 Replacing an inner node: the next intermediate steps. 121
4.25 The result of replacing an inner node. 122
4.26 The completeness of the compound transformations. 123
4.27 $\wedge$-linear normal form in the context of the decision procedure. . . 125
4.28 A running example: prior to linearising an $\wedge$-labelled connective. 126
4.29 A running example: after linearising an $\wedge$-labelled connective. . . 126
4.30 Linearising $\wedge$-labelled connectives. 127
4.31 An alternative order in which to linearise $\wedge$-labelled connectives. 127
4.32 The result of linearising all $\wedge$-labelled connectives in a diagram. . 128
4.33 A concrete example: prior to reducing the out-degree of n. 129
4.34 A concrete example: after reducing the out-degree of n by one. . . 130
4.35 A generalized diagram prior to pushing syntax. 132
4.36 A generalized diagram after pushing syntax. 132
4.37 A running example: $\wedge$-linear normal form. 133
4.38 A running example: prior to pushing syntax. 133
4.39 A running example: after pushing the first contour one step. . . . 134
4.40 Pushed syntax normal form in the context of the decision procedure.134
4.41 Preparing a diagram before pushing syntax, part 1. 135
4.42 Preparing a diagram before pushing syntax, part 2. 135
4.43 An application of *add missing zone*. 136
4.44 A generalized diagram in which we wish to push syntax forwards. 137
4.45 Pushing the first contour forwards. 138
4.46 Pushing contours forwards. 139
4.47 Pushing other syntax forwards. 140
4.48 Pushing shading from one unitary diagram to another. 140
4.49 Pushing an arrow from one unitary diagram to another. 141
4.50 Add contour components. 142
4.51 Pushing contours without $\wedge$-linear normal form. 144
4.52 Pushing contours with $\wedge$-linear normal form. 144
4.53 Preparing to push a spider into an shaded zone. 145
4.54 The result of pushing a spider into an shaded zone. 146

4.55 Inconsistency due to spiders and shading. 147
4.56 Preparing to push a spider into an unshaded zone. 149
4.57 The result of pushing a spider into an unshaded zone. 149
4.58 A running example: pushed syntax normal form. 163
4.59 The syntactic sub-diagram relationship. 164
4.60 A running example: disjunctive normal form. 168
4.61 Diagrammatic disjunctive normal form. 169
4.62 Disjunctive normal form in the context of the decision procedure. 170
4.63 Removing a non-linear $\vee$-labelled connective. 171
4.64 Intermediate steps in removing a nested non-linear $\vee$-labelled connective. 172
4.65 Multiple nested non-linear $\vee$-labelled connectives. 173
4.66 Relabelling a linear $\vee$-labelled node. 174
4.67 A generalized diagram with a root node labelled by the trivial diagram. 175
4.68 Pushing copies of a unitary diagram past the $\vee$-labelled node. . . 175
4.69 A generalized diagram in disjunctive normal form. 176
4.70 The algorithm which reduces linear generalized diagrams, in context.188
4.71 A running example: linear sub-diagrams in pushed syntax normal form. 189
4.72 The sub-diagram relationship. 190
4.73 A syntactic sub-diagram which is inconsistent due to the shaded zones condition. 191
4.74 A syntactic sub-diagram which is inconsistent due to the spiders' distinctness condition. 191
4.75 A syntactic sub-diagram which is inconsistent due to the spiders habitat condition. 191
4.76 An application of the *relabel sub-diagram* rule. 193
4.77 An application of the *relabel inconsistent sub-diagram* rule. 194
4.78 The algorithm to convert a generalized diagram to disjunctive normal form. 202

5.1 A unitary diagram and its γ-components. 205
5.2 A non-unitary generalized diagram. 206

5.3 A generalized diagram in disjunctive normal form. 207
5.4 Adding a contour so as to split every zone. 209
5.5 Adding a contour without splitting every zone. 209
5.6 Inconsistency arising from arrows sourced on universal spiders. . . 211
5.7 The complexity of the unitary decision procedure. 213
5.8 Inconsistent branches produced by pushing contours. 214
5.9 Stapleton's completeness proof strategy. 216
5.10 Shadeable zones. 217
5.11 Shadeable zones and derived contours. 218
5.12 A redundant arrow. 219
5.13 Redundancy inferred from a set of arrows. 219
5.14 Redundancy inferred from a set of arrows continued. 219
5.15 Redundancy and the sources of arrows. 220
5.16 Inferring redundancy from a subset of the source spiders. 221
5.17 Redundancy with no subset relation of source spiders. 221

Acknowledgements

I'd like to thank those in the diagrams community and elsewhere who have given me opportunities to discuss my work and who have reviewed that work, providing feedback which was often very valuable. Thanks to my supervisors, Gem and John, for their thoughtful guidance, their enthusiasm, and for helping to make my studies as enjoyable as they have been.

Thanks to David and Maureen for inspiration, and to Sarah, Lauren and Megan for keeping me going at home.

Declaration

I declare that the research contained in this thesis, unless otherwise formally indicated within the text, is the original work of the author. The thesis has not been previously submitted to this or any other university for a degree, and does not incorporate any material already submitted for a degree.

Signed:

Dated:

Chapter 1

Introduction

Modern logic has its roots in the work of Leibniz, who proposed the first logical calculus in his *characteristica universalis* [30]. Leibniz categorises the concerns of what would become formal logic as *ars inveniendi*, the art of finding, and *ars iudicandi*, the art of judging. In modern terms, the former corresponds to logical completeness, the ability to generate a complete set of theorems for a given language, while the latter corresponds to the decision problem: the search for a mechanical procedure to judge the satisfiability of a theorem. Modern logic began in earnest in the mid to late 19th century, following the work of Frege and Russell, and many of the first projects of its practitioners did indeed fall into one or another of Leibniz' two categories. Thus, the effort to identify decidable fragments of logical systems has a long tradition, with what is now a very substantial body of results, summarised and described in Börger et al. [2]. These results identify fragments of larger systems that are decidable, and fragments which are not.

This thesis is concerned with the decision problem in a visual, rather than a symbolic, setting. In it, we aim to develop *ars iudicandi tabula picta* – the art of judging diagrams. By contrast with symbolic logics, relatively little is known about which fragments of the more expressive diagrammatic logics are decidable. Clearly, any monadic diagrammatic logic (which includes only one place predicates) is decidable (such as Euler/Venn [47], Euler diagrams [17], spider diagrams [26] and Venn-II [36], all of which are described below). However, a visual logic capable of representing binary predicates, such as constraint diagrams, is potentially undecidable. As we will see, the constraint diagram notation includes elements called *arrows* and *spiders*, which allow users to describe binary relationships and to construct arbitrary quantifier alternations; permitting arbitrary quantifier alternations is a typical source of undecidability. A standard approach to creating a decidable fragment is to limit the number of quantifier alternations in some way.

In this thesis we will define a fragment of generalized constraint diagrams and show that this fragment is decidable by constructing a decision procedure. Constraint diagrams were proposed as a means of modelling software systems, with generalized constraint diagrams being a recent refinement. It is easy to see that the ability to determine the consistency of a model is a fundamental benefit to users of the system. Furthermore, the close study of the syntactic conditions of consistency in generalized constraint diagrams brings insights into the way in which syntactic elements interact. Since decision procedures exist for earlier versions of the constraint diagram notation, we use the decision procedure we develop as a means to focus on the novel syntactic features of the generalized case, and their effect on the expressiveness of the system.

We begin by surveying related work, focusing especially on constraint diagrams and the numerous related diagrammatic logics.

1.1 Related work

Euler [10] is credited with introducing diagrams which represent logical propositions. Equally, Euler diagrams can be considered to make assertions about sets. Figure 1.1(a) presents an Euler diagram which can be read in either of the following ways:

- All B is A. No A is C.
- $\forall x \big((B(x) \Rightarrow A(x)) \wedge \neg(A(x) \wedge C(x))\big)$.

Hammer provided Euler diagrams with a formal basis by presenting a sound and complete reasoning system based on Euler diagrams [17, 44]. An Euler diagram is a collection of closed curves called *contours* which represent sets, within an enclosing rectangle. Figure 1.1(a) shows an example with three contours, labelled A, B and C. Containment, intersection and disjointness are represented by the placement of contours, so the diagram asserts $B \subseteq A$ and $A \cap C = \emptyset$.

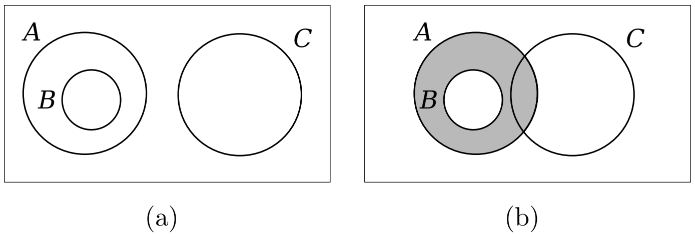


Figure 1.1: Two Euler diagrams.

A *zone* is a set of points in the diagram that can be described as being inside certain contours and outside all others. The diagram in Figure 1.1(b) has five zones; one inside A but outside B and C, one inside A and B but outside C, and so on. The region outside all contours but inside the boundary rectangle is also a zone. In Euler's original notation, each zone is necessarily non-empty. Hammer added the syntactic device of shading within a zone, which asserts the emptiness of the set represented by that zone. There are two shaded zones in the diagram in Figure 1.1(b), and the fact that these zones are shaded asserts that $A \cap C = \emptyset$ and $A - B = \emptyset$. Reasoning is carried out by the application of rules which transform one diagram into another. A proof consisting of Euler diagrams is formed by applying inference rules repeatedly to transform an initial diagram (the premise) into the target diagram (the conclusion). Hammer [17] defined three inference rules, including one which adds a missing zone to a diagram, and one which adds a missing contour to a diagram. In Figure 1.2 we can use the add missing zone rule to the zone inside A and B to d_1, giving d_2. This rule is sound, since both missing and shaded zones represent the empty set. We can use the rule which adds a contour to add C to d_2, giving d_3. We say that this rule 'maintains' the shading of the original diagram; the zone inside A and B is shaded in d_2, whilst in d_3 the two zones inside A and B are shaded.

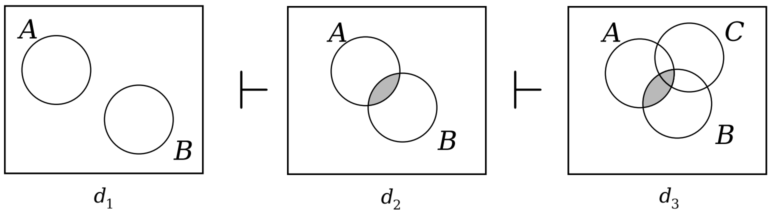


Figure 1.2: A proof using Euler diagrams.

Venn noted several limitations of Euler diagrams related to their strictness [49]. Perhaps the most telling of these is the fact that, since zones necessarily represent non-empty sets in Euler's original notation, we cannot use an Euler diagram to represent sets A and B if we do not know whether the intersection of A and B is populated. Venn aimed to formalise and generalise the use of diagrams in logic, developing a notation in which each intersection between the represented sets is shown without making any claims about the existence of elements. In Venn's diagrams the emptiness of a zone is indicated by shading, rather than the absence of the zone, as in Euler's original notation. Figure 1.3 shows a Venn diagram whose meaning is equivalent to that of the Euler diagram in Figure 1.1(a), except that the non-shaded zones in Figure 1.3 may or may not be empty.

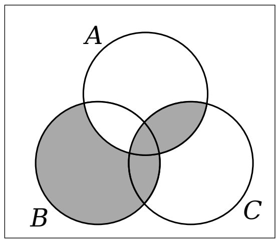


Figure 1.3: A Venn diagram.

Charles Peirce [31] added syntax to Venn diagrams to make them more expressive, allowing for existential statements and disjunction. Peirce made a crucial advance towards formal diagrams by developing, for the first time, *rules of transformation*, such as those described above for Euler diagrams, used to draw valid inferences from diagrams. In Peirce's diagrams, emptiness is depicted by the symbol O. An X placed in a zone indicates that the set represented by the zone contains an element. Zones containing neither O nor X symbols may or may not be empty. The O and X symbols may be joined by a straight line to indicate disjunction. So, in Figure 1.4 the set represented by the zone inside C and outside A and B is either empty or contains an element.

The lack of distinction between the syntax of these rules and their semantic meaning restricted the work that could be done in the metatheory of Peirce's diagrams. Peirce suspected, for example, that the set of rules defined was incomplete, but was unable to prove that fact. Indeed, it was only after establishing separate syntax and semantics for a notation adapted from Peirce's work that

Shin [37] was able to show that Peirce's set of rules is incomplete.

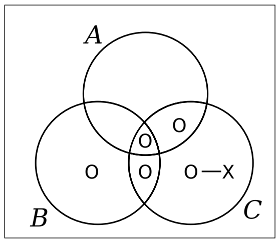


Figure 1.4: A Peirce diagram.

Fully formalised visual logics emerged in the 1990s, beginning with Shin's [36] work on the Venn-I and Venn-II reasoning systems. Shin adapted Venn-Peirce diagrams by using shading to represent emptiness, as in Venn diagrams, and by enclosing each diagram in a boundary rectangle to represent the universal domain.

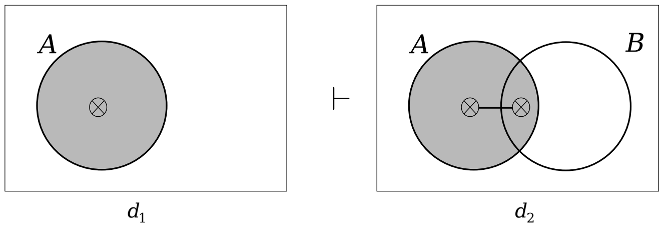


Figure 1.5: Venn-I.

Venn-I is a simplification of Peirce's diagrams. Shin provides six valid rules of inference for the system. Figure 1.5 illustrates the rule which introduces a new contour to a diagram. We can use the rule to introduce a new contour, B, to the diagram d_1, resulting in diagram d_2. From d_1 we know there is a single element in the set represented by A. Since we do not know whether that element is also a member of B, d_2 asserts that the element is either in A and not B, or in both A and B. Furthermore, the inference rule maintains the shading from d_1, by ensuring that the zone inside both A and B is shaded in d_2. Shin makes a clear distinction between diagrammatic syntax and its underlying semantics, a distinction which had been imperfectly made in earlier systems. The semantics is model-theoretic, in which an interpretation of a diagram is a mapping from curves, or contours, to subsets of a universal domain. Venn-II increases the expressiveness of Venn-I by allowing so-called *unitary* diagrams (of the type we have seen so far) to be connected with a straight line, indicating disjunction.

Figure 1.6 gives an example which states, among other things, that either there is an element in $C - (A \cup B)$, or C is empty.

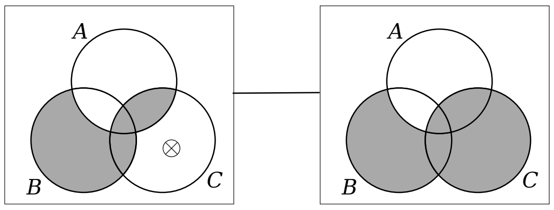


Figure 1.6: Venn-II.

Venn-II is equivalent in expressive power to monadic first order logic [36]. Shin's influential work on Venn-I and Venn-II was part of a growing interest in visual logics based on the work of Euler and Venn in the 90s, which is surveyed in Stapleton [39]. Swoboda [47] adapted Venn-II to produce Euler/Venn diagrams. The Euler/Venn system uses Euler diagrams as the underlying notation, rather than the more restrictive Venn diagrams (as in Venn-II) and introduces named constants.

Spider diagrams [13, 23, 21, 24, 26] extend Euler diagrams with shading by adding syntax to represent individual elements in various ways. Instead of the $\otimes$-sequences found in Venn-II, *spiders*, which are collections of round or rectangular dots joined by straight lines, are used to represent the existence of elements. A spider may have several *feet*, which are dots placed in distinct zones and joined by straight lines. Similarly to the $\otimes$-sequences of Venn-II, a spider with several feet provides disjunctive information, asserting the existence of an element in exactly one of the zones in which its feet are placed. Unlike the $\otimes$ notation in Venn-II, distinct spiders necessarily represent distinct elements.

Figure 1.7 shows a spider diagram. There is a spider with a single foot in the zone inside A but outside B and C, indicating that there is at least one element in the set represented. There are two spiders in the zones inside B and C, one of which has three feet. The shading in the zone inside C but outside A and B places an upper bound on the cardinality of the set the zone represents – the shading asserts that the set is empty other than for the elements represented by spiders placed in the zone. Similarly, spiders place a lower bound on the cardinality of the set represented by the zone in which they are placed. Thus, in Figure 1.7, the set represented by the zone inside C but outside A and B contains either exactly one or exactly two elements.

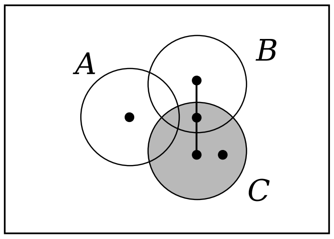


Figure 1.7: A spider diagram.

In [28], Molina developed the first sound and complete system of spider diagrams, called SD1. In SD1, spiders cannot be placed in shaded zones. This restriction is removed in SD2 [28], and the diagram in Figure 1.7 is an SD2 diagram.

The full formalisation of diagrammatic logics depends on a clear separation between a notation and its underlying meaning. Howse et al. [19] explore a parallel distinction within the syntactic level which is beneficial to providing a clear metatheory for a visual logic; this is the distinction between *type* and *token* syntax. It is easier to make mathematically rigorous statements about a diagrammatic notation if we are able to ignore its semantically unimportant details. These details may include the shape of diagrammatic elements, or their relative position on page or screen. For instance, in Figure 1.7 the relative size of the circles A, B and C tells us nothing about the size of the sets they represent, nor does it convey any other semantic information. The size of the circles we draw when creating Euler or spider diagrams is an important consideration in terms of the readability of the diagram, but not in terms of its formal meaning (at least, so far as this work is concerned). For this reason, it is important to consider the *abstract* syntax (called the type syntax in [19]) of a notation separately from that notation's *concrete* (or token) syntax. The concrete syntax describes the spatial conditions of a diagram as it is drawn, including such constraints as we may wish to place on presentation; the abstract syntax is a notation which formalises that which is essential for a notation to have unambiguous meaning, and it is this notation which is used in definitions of semantics and metatheory. The diagrammatic logics described in this section each use their own abstractions but all are based on the idea of representing diagrams as tuples of sets of labels and identifiers of other diagrammatic elements to represent the essential information conveyed by a diagram.

1.1.1 Constraint diagrams

In parallel to the mathematical interest in diagrammatic reasoning, there has been a constant need for structured diagrams in software development. Since its inception, this field has made use of a wealth of notations to describe, for instance, structural schema, functional constraints and process flow. Examples of notations widely used by software engineers include Harel statecharts [18], entity relationship diagrams [5] and the Unified Modelling Language (UML) [48]. Whilst such notations must be considered effective tools of communication since they are so widely used in the software process, they are not, in general, suitable for precise specification. The UML, for example, is a collection of several disparate notations with no formal semantics, and makes use of an extra-diagrammatic symbolic notation to describe operational constraints [29]. Although much work has been done towards formalising UML (for example, [6]), formal fragments of UML require non-diagrammatic symbolic components in order to specify operational constraints and are not developed into axiomatic systems with rules of inference.

Thus, traditional formal methods such as Z [32] and Coq [1] remain the only practical choice when verifying critical software. As Sheard [34] states, this situation maintains a 'semantic gap' between programming and verification, since the tools and techniques used to perform each activity are dissimilar. A recent approach to precise diagrammatic specification is found in the diagram predicate framework [33], in which UML class diagrams are given a formal, categorical semantics. Based as it is on UML, this approach is not purely diagrammatic and requires a textual component to specify constraints.

Constraint diagrams were introduced by Kent [27] with the purpose of formally modelling software systems, and are sufficiently expressive to specify operational contracts [25]. Kent's concern was with providing a fit-for-purpose, highly expressive diagrammatic notation that better fitted with the other visual notations on offer to the software engineer. Constraint diagrams extend the notation of spider diagrams by introducing syntax to represent universal quantification and binary relations. An example, taken from [25], can be seen in Figure 1.8. This diagram 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 asterisk is a so-called universal spider which acts as a universal quantifier, the arrows allow us to make statements about binary relations and the closed curves represent sets (or classes).

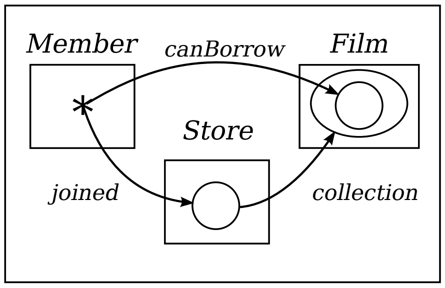


Figure 1.8: A constraint diagram.

The study and development of constraint diagrams has led to an improved understanding of what can be expressed in a visual, yet formal, manner. Indeed, the process of formalising Kent's original notation revealed the difficulty of developing highly expressive diagrammatic notations in which arbitrary quantifier alternations can occur [14]. In particular, constraint diagrams as originally presented are ambiguous; since spiders represent quantifiers, the order in which they are read affects the meaning of the diagram. In the absence of information about the scope of quantifiers (spiders), a diagram may have multiple incompatible interpretations. The diagram in Figure 1.8 is unambiguous. In Figure 1.9, the same diagram is altered by the addition of a so-called existential spider, which is represented by a solid circular dot and asserts the existence of an element in the zone in which the spider is placed. Two possible meanings can be ascribed to this diagram, depending on the order in which we read the spiders, which are labelled x and y for convenience (spider labels are not part of the constraint diagram notation). If we read the universal spider, x, first, then the meaning of the diagram includes the assertion that, for each member, x, there is a film, y, that x can borrow. If we read the existential spider, y, first, then the meaning includes the assertion that there is a certain film, y, that every member can borrow.

To overcome this problem, Fish et al. [12] proposed *augmented* constraint

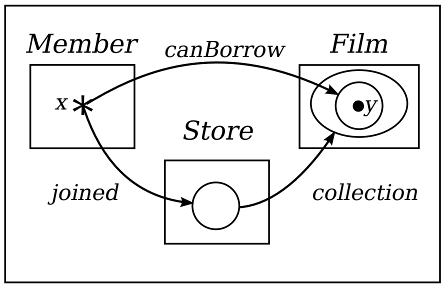


Figure 1.9: An ambiguous constraint diagram.

diagrams. These are constraint diagrams augmented by *reading trees*, extra-diagrammatic graphs which provide a reading order for the spiders of the diagrams. The two possible reading trees for the diagram in Figure 1.9 are $\overset{*}{x}\rightarrow\overset{\bullet}{y}$ and $\overset{\bullet}{y}\rightarrow\overset{*}{x}$. Reading trees need not be linear and may provide more complex quantifier scoping by the use of bracketing information. Figure 1.10 shows a augmented constraint diagram and its reading tree. The reading tree is non-linear and imposes bracketing information on the meaning assigned to the diagram, with the result that x is not in the scope of y or z.

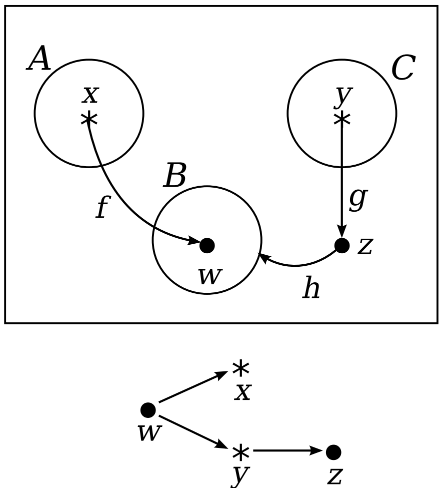


Figure 1.10: An augmented constraint diagram.

In contrast to the extra-diagrammatic reading tree, generalized constraint diagrams [40] impose a reading order as an implicit feature of the notation. Figure 1.11 shows a generalized diagram that expresses the same information as the diagram in Figure 1.9 augmented by the reading tree $\overset{*}{x}\rightarrow\overset{\bullet}{y}$. The diagram D is composed of two unitary diagrams, labelled d_1 and d_2, connected by an arrow. The reading order is implied by the fact that the diagram is 'read' from left to right, and thus d_2 is 'in the scope of' d_1. Since the spider y appears in d_2 but not in d_1, when the meaning of D is constructed, y is read after, and is in the scope of, x. We can consider, informally, the meaning of D to be given by $d_1 \wedge d_2$, although this is potentially misleading since, in this case $d_1 \wedge d_2 \not\equiv d_2 \wedge d_1$, due to the interaction of the quantifiers x and y. To maintain unique meanings, constraints are placed on the way in which spiders can be added to the diagram; if a unitary diagram, d, contains a universal spider which does not appear in any of the 'ancestors' of d (those unitary diagrams that appear before d) then d cannot contain any other 'new' spiders.

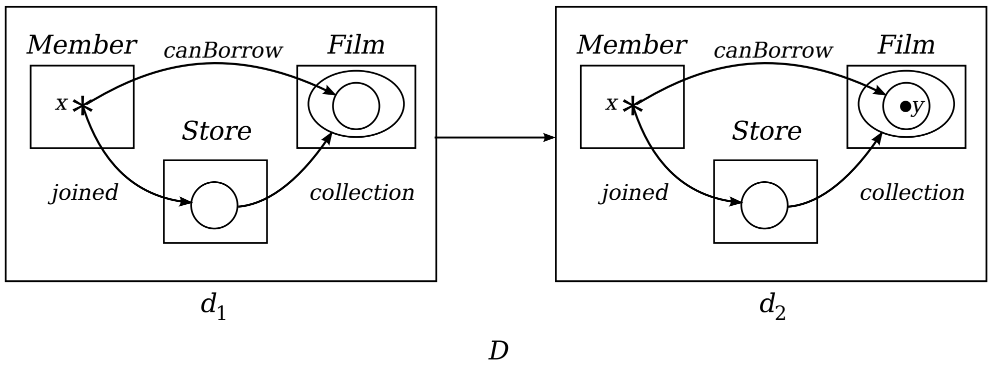


Figure 1.11: A generalized constraint diagram.

The tree structure of generalized constraint diagrams may contain branches, representing disjunction and conjunction. In the diagram in Figure 1.12, the vertical bar represents disjunction, and so the meaning of the diagram is, informally, $d_1 \wedge (d_2 \vee d_3)$. In Figure 1.13, the diagram from Figure 1.12 is extended with a conjunctive branch, represented by the fork attached to d_2. Informally, we could say that the meaning is now $d_1 \wedge (d_3 \vee (d_2 \wedge (d_4 \wedge d_5)))$, although this informality is potentially misleading due to the issues of quantifiers and their scope discussed above[1].

[1]In fact, issues of scope do not come into play in this case. Since Figure 1.13 contains only existential spiders, the order in which the quantifiers they represent are read does not affect the meaning of the diagram.

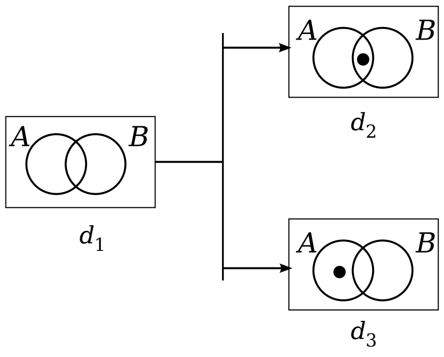


Figure 1.12: Disjunction in generalized constraint diagrams.

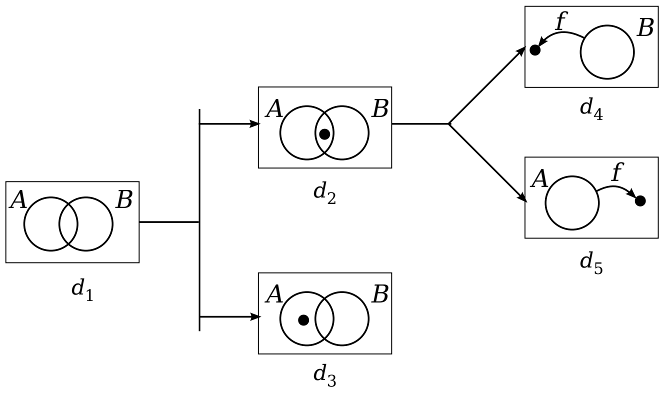
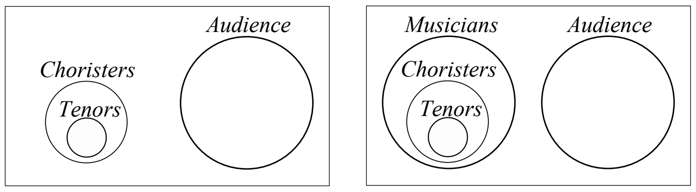


Figure 1.13: Disjunction and conjunction in generalized constraint diagrams.

Part of the motivation for developing constraint diagrams lies in the claim that their use makes it easier to create and comprehend formal specifications, for some group of software engineers, than traditional formal methods. Creating a formal specification using a system such as Coq, and showing that a program conforms to that specification, requires specialised skills possessed by relatively few programmers. In contrast, diagrams are widely used and accepted in the software process. Arguably, Euler diagrams, the notation underlying constraint diagrams, are intuitive; that is, they are easy to use and possess natural affordances which can be classified in several ways. An intuitive diagram has the power to convey information efficiently and in such a way that its message is easily understood. A diagram with these attributes is said to be *well-suited* or *well-matched* to its subject [16, 35]. Regarding efficient representation in diagrams, Shimojima [35] identified *free rides* as phenomena which occur when a diagram provides valid information as an unintended consequence of its physical conditions. For example, Figure 1.14(a) shows two Euler diagrams. In Figure 1.14(a), the fact that the circle *Tenors* is placed within *Choristers* tells us that all tenors are choristers. Since the circles *Choristers* and *Audience* do not overlap, the diagram states that no choristers are in the audience. The viewer can also infer, as a free ride, that no tenors are in the audience. Adding the circle *Musicians* to the diagram in Figure 1.14(a) to produce the diagram in Figure 1.14(b) asserts that all choristers are musicians and that no musicians are in the audience, but also that all tenors are musicians, as a second free ride. Here we see an example of well-matchedness since the transitive property of the subset relation is mirrored in the diagram by the containment of circles.



(a) (b)

Figure 1.14: Free rides

The usability of constraint diagrams, as distinct from the underlying Euler notation, requires further study. In [11], Fetais and Cheng conducted an experimental study in which users performed a comprehension task using either constraint diagrams or natural language, and found no significant difference in performance between the two groups. Generalized constraint diagrams (GCDs) were designed with usability and cognitive principles in mind. The usability of GCDs was addressed in their original presentation [41], in which Stapleton and Delaney state five principles that guided the design of the notation:

1. *Well-matchedness principle*: that syntactic relations mirror semantic ones.
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.

5. *Construction principle*: impose only the necessary restrictions on what constitutes a well-formed statement [41].

The authors argue that GCDs satisfy the principles by identifying notational features in each case. For example, GCDs satisfy the well-matchedness principle since, arguably, the underlying Euler diagram notation is well-matched, as is the use of arrows to represent properties of binary relations which are, of course, a directional relationship. Furthermore, GCDs satisfy the explicitness principle by virtue of the way in which meaning is assigned to a diagram. For instance, the meaning of spiders is explicitly provided; spiders quantify over the information in the unitary diagram, d, in which they first appear, and the information in the descendants of d [41]. The design of GCDs is studied further in Coppin et al. [8], in which the authors argue that the notation suits the cognitive requirements which arise when communicating proofs. More research is required to analyse the usability of constraint diagrams and their generalized case in comparison with related notations.

It has been shown that generalized constraint diagrams form an undecidable logic, being at least as expressive as first order logic containing arbitrary one and two place predicates [41]. This expressivity allows us to create diagrams with only infinite models, for which no decision procedure can exist. For instance, Figure 1.15 shows a generalized constraint diagram which requires an infinite model, expressing a property of the integers. A translation of the semantics of this diagram to first order logic yields the following sentence:

$$\forall x(\mathbb{Z}(x) \Rightarrow \exists y(\mathbb{Z}(y) \wedge x \neq y \wedge f(x, y))).$$

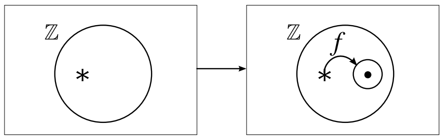


Figure 1.15: An undecidable generalized constraint diagram.

We identify a decidable fragment of generalized constraint diagrams by excluding universal spiders, and call the resulting fragment the *existential fragment* (EF).

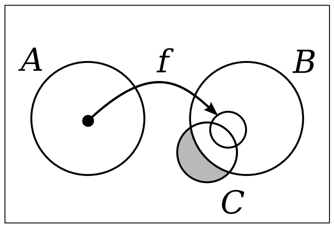


Figure 1.16: A diagram from the existential fragment.

Although the existential fragment does not contain universal spiders, this does not imply that it lacks universal quantification: the use of shading and arrows gives rise to statements that are universally quantified. For example, translating the semantics of the diagram in Figure 1.16 to first order logic yields the following sentence, which includes universal quantifiers:

$$\exists x(A(x) \wedge \forall y(f(x,y) \Rightarrow B(y)) \wedge \forall z((C(z) \Rightarrow B(z)) \wedge (A(z) \Rightarrow \neg B(z)))).$$

By removing universal spiders, we restrict the manner in which quantifiers can alternate, with the result that all existential quantification comes before all universal quantification. As we describe in the next section, this method, in which a certain quantifier prefix is specified, is closely related to the way in which decidable fragments of symbolic logics are identified.

1.1.2 The decision problem in symbolic and visual logic

The search for decidable fragments of logical systems is part of the wider study of computability, and arose from the series of problems posed by Hilbert in 1900 [7]. Particular fragments of a logical system, known as languages, are categorised along four axes:

1. the quantifier prefix sentences drawn from the language may have, after conversion to prenex normal form,

2. the arities of predicate and function symbols drawn from the language,

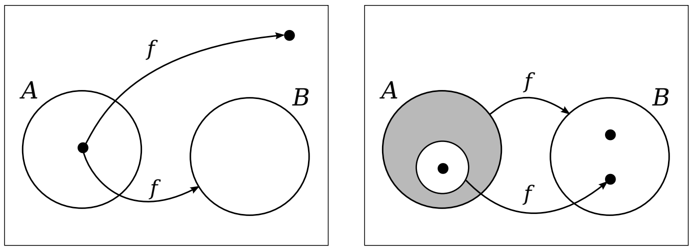
3. the existence of an equality relation, and

4. the existence of a finite model for any sentence drawn from the language.

For instance, Ramsey [2] showed in 1930 that the first order language with prefix $\exists^*\forall^*$, with two-place predicates, finite models and equality, is decidable. For languages with finite models, many proofs of decidability hinge on the construction of a minimal model for the language, called the canonical, or Herbrand model [2, p26]. Although the decision problem for first order symbolic logic is solved, open questions include the exact complexity of the decision problems for particular languages [2, 7].

In this section we will present a high-level view of the decision problem in diagrammatic reasoning, before examining the particular case of the decision problem for monadic languages (symbolic and diagrammatic) in more detail in section 1.1.3. Several of the diagrammatic logics described in the previous section are trivially decidable, and the decision problem does not become interesting in diagrammatic reasoning until we consider more expressive systems such as constraint diagrams. In [43] Stapleton et al. develop a decision procedure for a sound and complete system based on a decidable fragment of (non-generalized) constraint diagrams. The decision procedure is extracted from the proof of completeness, which works by the algorithmic manipulation of a diagram into its *maximal form* by such operations as adding syntactic elements and splitting unitary diagrams into disjunctions of simpler diagrams. The decision procedure works by the pairwise comparison of the arrows of a diagram. Adopting the canonical models used in decidability proofs in symbolic logic, the proof of the link between syntactic conditions and satisfiability proceeds by showing that each satisfiable diagram has a finite, minimal model. For constraint diagrams, the canonical model, called the standard model, is one in which the universe is given by the set of spiders in the diagram [38, p236].

Figure 1.17 shows two constraint diagrams, neither of which requires a reading tree in order to have an unambiguous interpretation. In the (non-generalized) constraint diagram in Figure 1.17(a), the arrow sourced on the spider inside A and which targets the spider in the zone outside A and B tells us, informally, that when the domain of f is restricted to the element represented by the spider inhabiting A, the relational image is the element represented by the spider inhabiting the zone outside of A and B. However, the arrow sourced on the same spider and which targets the contour B provides contradictory information: when the domain of f is restricted to the element represented by the spider inhabiting A, the relational image of f is B. In Stapleton's terminology, the two arrows in the diagram are *pairwise incompatible.*



(a) (b)

Figure 1.17: Consistency in constraint diagrams and the generalized case

The diagram in Figure 1.17(b) is a generalized constraint diagram. The unlabelled contour inside A is called a *derived* contour, and represents an arbitrary subset of the set represented by A. From the arrow sourced on the derived contour and from the shading in the zone inside A but outside the derived contour, we can infer that, when the domain of f is restricted to A, the image of f is a proper subset of B. The arrow sourced on A provides contradictory information, i.e. that when the domain of f is restricted to A, the image of f is B. The notion of pairwise compatibility does not capture this inconsistency, which arises because generalized constraint diagrams, unlike earlier versions of constraint diagrams, contain arrows sourced on contours. In chapter 3, we provide examples of inconsistency in generalized unitary diagrams which show that derived contours and arrows sourced on contours interact in other ways, justifying a need for more general definitions.

Like the fragment of generalized constraint diagrams presented in this thesis, the fragment of constraint diagrams presented in [43] excludes universal spiders. Semantically, there are numerous differences between the two systems, the key one being that generalized constraint diagrams are given a second order semantics (see section 2.3). Syntactically, there are two key differences between the unitary diagrams of [43] and those of the EF. First, all arrows are sourced on spiders in (non-generalized) constraint diagrams, leading to a relatively straightforward definition of consistency and, hence, satisfiability. Secondly, there are no derived contours in [43], which reduces expressiveness. GCDs include both derived contours and arrows sourced on (given or derived) contours; as we will see in section 3.1, this has a significant impact on the notion of consistency for generalized diagrams.

1.1.3 Decidability in monadic languages

In this section we compare approaches to the decision problem in two monadic languages. The first of these, monadic first order logic with equality, denoted $\mathcal{MFOL}_=$, is a symbolic logic, whilst the second, spider diagrams, is diagrammatic. In [46] Stapleton et al. show that these two systems are equivalent in expressive power. Definitions 1.1.1 and 1.1.2 are taken from [46]. To begin, monadic predicate symbols and variables are drawn from the countably infinite sets $\mathcal{P}$ and $\mathcal{V}$, respectively.

Definition 1.1.1. The first order language $\mathcal{MFOL}_=$ consists of the following:

1. **Atomic formulae** which are defined as follows:
   (a) if x_i and x_j are variables then $(x_i = x_j)$ is an atomic formula,
   (b) if $P_i \in \mathcal{P}$ and x_j is a variable then $P_i(x_j)$ is an atomic formula.
2. **Formulae**, which are defined inductively:
   (a) Atomic formulae are formulae.
   (b) $\bot$ and $\top$ are formulae.
   (c) If p and q are formulae so are $(p \wedge q)$, $(p \vee q)$ and $\neg p$.
   (d) If p is a formula and x_j is a variable then $(\forall x_j\, p)$ and $(\exists x_j\, p)$ are formulae.

We define $\mathcal{F}$ and $\mathcal{S}$ to be the sets of formulae and sentences (formulae with no free variables) of the language $\mathcal{MFOL}_=$ respectively.

Since we can construct the universal closure of any formula, F, giving a sentence, S [2, p25], we can restrict consideration to sentences. Sentences in $\mathcal{MFOL}_=$ are interpreted in the standard way, via *structures*, with the only exception to the standard approach being that structures with an empty domain are permitted.

Definition 1.1.2. We define $m = (U, =_m, P_1^m, P_2^m, \dots)$ to be a **structure**, where m satisfies the following:

1. U is a set,
2. $=_m$: $U \times U$ is the equality relation on U, and
3. P_i^m is the interpretation of the monadic predicate symbol P_i in the structure m (that is, $P_i^m \subseteq U$).

A concrete decision procedure for $\mathcal{MFOL}_=$ is given in [2, p25], and is due to Büchi. We will describe the approach informally: given a sentence S, we produce a domain, U, consisting of the variables and predicate symbols of S, and call this the canonical domain. Then the canonical model for S is one in which the variables and predicate symbols of S are interpreted as themselves in the domain U. Finally, Büchi showed that S is satisfiable if and only if we can construct its canonical model.

Similarly to the above results for $\mathcal{MFOL}_=$, the decidability of spider diagrams can be shown by identifying a finite, minimal model for each diagram. In [28] Molina provides a procedure for constructing a standard model for a satisfiable spider diagram, which takes the set of spiders as the universal domain; he then showed that if we are unable to construct the standard model for a diagram, d, then d is unsatisfiable, thus providing a decision procedure for spider diagrams. To demonstrate this we will briefly describe the abstract syntax and semantics of spider diagrams. The following definitions and theorem 1.1.1 are taken from [26]. In this presentation, the 'underlying' diagram of a spider diagram is an Euler diagram, unlike systems SD1 and SD2, whose underlying diagrams are Venn diagrams.

As previously stated, the theory of spider diagrams works at the level of abstract syntax, ignoring many superfluous details of drawn diagrams. To describe a spider diagram in the abstract syntax we need to list the contour labels of the diagram, its zones, those zones which are shaded, and the spiders of the diagram. Contour labels are drawn from a countably infinite set $\mathcal{L}$. Howse et al. [26] denote the set of all finite subsets of a set S by $\mathbb{F}S$. A zone is a pair of sets of contour labels, $z = (in, out)$, where the labels in in are inside z and those in out are outside z. Figure 1.18 shows a spider diagram, d, with the following four zones:

1. $(\emptyset, \{A, B\})$,
2. $(\{A\}, \{B\})$,
3. $(\{A, B\}, \emptyset)$, and
4. $(\{B\}, \{A\})$.

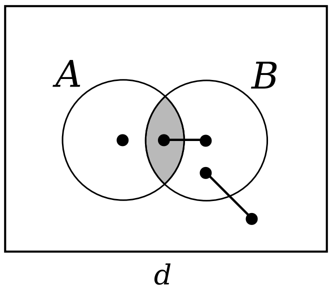


Figure 1.18: Zones and spiders in spider diagrams.

Definition 1.1.3. A **zone** with labels in $\mathcal{L}$ is an ordered pair (in, out) where $in, out \subseteq \mathbb{F}\mathcal{L}$ and $in \cap out = \emptyset$. Define $\mathcal{Z}$ to be the set of zones on $\mathcal{L}$,

$$\mathcal{Z} = \{(in, out) \in \mathbb{F}\mathcal{L} \times \mathbb{F}\mathcal{L} : in \cap out = \emptyset\}.$$

If $z = (a, b) \in Z$ then the set $a = c(z)$ is called the set of contour labels that **contain** z. A **region** with labels in $\mathcal{L}$ is a set of zones; $\mathcal{R} = \mathbb{P}\mathcal{Z}$ denotes the set of **regions** on $\mathcal{L}$.

The habitat of a spider is the region in which its feet are placed. In Figure 1.18, there are spiders with habitat $\{(\{A\}, \{B\})\}$, $\{(\{A, B\}, \emptyset), (\{B\}, \{A\})\}$ and $\{(\{B\}, \{A\}), (\emptyset, \{A, B\})\}$. Several authors [20, 28] have given the abstract description of the set of spiders of a diagram by a set of identifiers and a habitat function (as, in fact, we do in chapter 2). In [26], Howse et al. choose instead to represent spiders by listing the number of spiders inhabiting each region. If there are $n > 0$ spiders inhabiting region r, then (n, r) is a *spider identifier*. The result of this choice is that each drawn spider diagram has a unique abstract description.

Definition 1.1.4. A **unitary spider diagram** with labels in $\mathcal{L}$ is a tuple $d = (L, Z, Z^*, SI)$ whose components are defined as follows.

1. $L = L(d) \in \mathbb{F}\mathcal{L}$ is a finite set of **contour labels**.
2. $Z = Z(d) \subseteq (in, L - in) : in \subseteq L$ is a set of **zones** $(Z(d) \subseteq \mathcal{Z})$ such that
   (a) for all $l \in L$ there exists $(in, L - in) \in Z$ such that $l \in in$, and
   (b) $(\emptyset, L) \in Z$.

   We define $R = R(d) = \mathbb{P}Z - \{\emptyset\}$ to be the set of **regions**.
3. $Z^* = Z^*(d) \subseteq Z$ is the set of **shaded zones**.
4. $SI = SI(d) \subset \mathbb{Z}^+ \times R(d)$ is a (finite) set of **spider identifiers** such that

$$\forall (n_1, r_1), (n_2, r_2) \in SI(r_1 = r_2 \Rightarrow n_1 = n_2).$$

   If $(n, r) \in SI$ we say there are n spiders whose habitat is r.

Additionally, the diagram $\perp = (\emptyset, \emptyset, \emptyset, \emptyset)$ is a unitary spider diagram.

We will illustrate the abstract syntax using Figure 1.19, which is taken from [26]. This diagram has the abstraction (L, Z, Z^*, SI), which is determined as follows:

1. The set of contour labels, L, is equal to $\{A, B, C\}$.
2. The set of zones, Z, is equal to
   (a) $(\{A\}, \{B, C\})$,
   (b) $(\{A, B\}, \{C\})$,
   (c) $(\{C\}, \{A, B\})$, and
   (d) $(\emptyset, \{A, B, C\})$.
3. The set of shaded zones, Z^*, is equal to $\{(\{A\}, \{B, C\})\}$.
4. The set of spider identifiers, SI, is equal to

$$\{(2, \{(\{A\}, \{B, C\}, (\{A, B\}, \{C\}))\}), (1, \{(\{C\}, \{A, B\})\})\}.$$

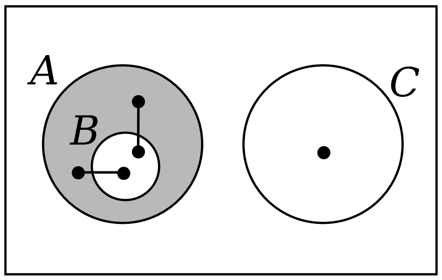


Figure 1.19: The abstract syntax of spider diagrams.

Definition 1.1.5. Let d be a unitary spider diagram.

1. If $(n, r) \in SI(d)$ then the region r contains n spiders which we denote $s_1(r)$, $s_2(r)$, ... , $s_n(r)$. We define $S(d)$ to be the set of all spiders in d:

$$S(d) = \{s_i(r) : (n_r, r) \in SI(d) \wedge 1 \leq i \leq n_r\}.$$

   The habitat mapping $\eta : S(d) \to R(d)$ is given by $\eta(s_i(r)) = r$ and we say that the spider $s_i(r)$ has **habitat** $\eta(s_i(r))$.

2. Let r be a region of d. The set of complete spiders **inhabiting** r in diagram d is:

$$S(r, d) = \{s \in S(d) : \eta(s) \subseteq r\}.$$

   The set of spiders **touching** region r in diagram d is

$$T(r, d) = \{s \in S(d) : \eta(s) \cap r = \emptyset\}.$$

   For any region r not in $R(d)$ we define $S(r, d) = \emptyset$ and $T(r, d) = \emptyset$.

Unitary spider diagrams can be combined using disjunction and conjunction. In Figure 1.20 shows a compound diagram, D, which is formed of two unitary spider diagrams, d_1 and d_2, in disjunction. The meaning of D is given by the disjunction of the meanings of d_1 and d_2.

In the next definition we simplify the presentation of [26], which includes details not relevant to our purpose such as multi-diagrams, which are the conjunction or disjunction of compound diagrams.

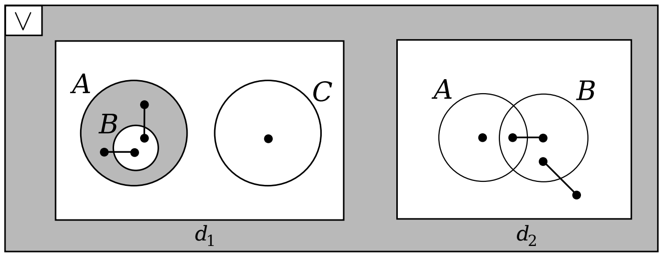


Figure 1.20: A compound spider diagram.

Definition 1.1.6. An **abstract spider diagram** is defined as follows.

1. Any unitary diagram is a spider diagram.

2. If D_1 and D_2 are spider diagrams then $D_1 \vee D_2$ is a spider diagram.

3. If D_1 and D_2 are spider diagrams then $D_1 \wedge D_2$ is a spider diagram.

For full details of the abstract syntax of spider diagrams, see [26]. Next, we consider semantics. Contour labels, zones and regions represent sets and subsets of a universal domain, U, while spiders assert the existence of elements in the region in which they are placed. The number of elements in the set represented by a shaded region is less than or equal to the number of spiders touching that region. This allows us to place lower and, in the case of shaded regions, upper bounds on the cardinality of the sets represented. Missing zones represent the empty set.

Definition 1.1.7. An **interpretation** is a pair, $m = (U, \Psi)$, where U is a set and $\Psi : L \to \mathbb{P}U$ is a function mapping contour labels to subsets of U. We extend Ψ to a set assignment to zones, $\Psi : Z \to \mathbb{P}U$. The set denoted by a zone, $z = (in, out)$, is defined to be the intersection of the sets denoted by the contour labels in in and the intersection of the complements of the sets denoted by the contour labels b:

$$\Psi(in, out) = \bigcap_{l \in in} \Psi(l) \cap \bigcap_{l \in out} \overline{\Psi(l)},$$

where $\overline{\Psi(l)} = U - \Psi(l)$. We also define $\bigcap_{l \in \emptyset} \Psi(l) = U = \bigcap_{l \in \emptyset} \overline{\Psi(l)}$. Finally we extend Ψ to a set assignment to regions, $\Psi : R \to \mathcal{P}U$. The set denoted by a region, r, is the union of the sets denoted by the zones which r contains:

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

We also define $\Psi(\emptyset) = \bigcup_{z \in \emptyset} \Psi(z) = \emptyset$.

As an example of the process of assigning a meaning to a spider diagram, consider Figure 1.21. Let $I = (U, \Psi)$ be an interpretation, where $U = \mathbb{N}$ and Ψ satisfies the following:

1. $\Psi(A) = \{1, 2\}$,
2. $\Psi(B) = \{2, 3\}$.

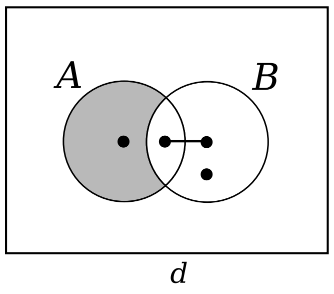


Figure 1.21: Interpreting a spider diagram.

Then we say that I *satisfies* d. We can see that, under I, the spider with two feet in d, whose habitat is in B, represents an element in $\Psi(A) \cap \Psi(B)$. In [26], as in earlier systems [28], the question of whether an interpretation satisfies a unitary spider diagram is captured by a set of conditions called the *semantics predicate.*

Definition 1.1.8. Let D be a diagram and let $m = (U, \Psi)$ be a set assignment to regions. We define the **semantics predicate**, denoted $P_D(m)$, of D. If $D = d$ $(\neq \perp)$ is a unitary diagram then $P_d(m)$ is the conjunction of the following three conditions.

1. **Distinct Spiders Condition.** The cardinality of the set denoted by a region r of a unitary diagram d is greater than or equal to the number of

complete spiders in r:

$$\bigwedge_{r \in R(d)} |\Psi(r)| \geq |S(r,d)|.$$

2. **Shading Condition.** The cardinality of the set denoted by a shaded region r of a unitary diagram d is less than or equal to the number of spiders touching r:

$$\bigwedge_{r \in R^*(d)} |\Psi(r)| \leq |T(r,d)|.$$

3. **Plane Tiling Condition.** All elements fall within sets denoted by the zones of d:

$$\bigcup_{z \in Z(d)} \Psi(z) = U.$$

If $D = \bot$ then $P_D(m) = \bot$. If $D = D_1 \vee D_2$ then $P_{D_1}(m) \vee P_{D_2}(m)$. If $D = D_1 \wedge D_2$ then $P_{D_1}(m) \wedge P_{D_2}(m)$.

Next, Howse et al. state the following theorem, which shows the decidability of spider diagrams.

Theorem 1.1.1: Every unitary spider diagram ($\neq \bot$) has a model.

Sketch. Given a non-false unitary diagram d, we follow the approach adopted by Molina [28] to construct a *standard model* for d, as follows. Take the universal set to be the set of spiders in d: $U = S(d)$. For each spider $s \in S(d)$, choose a zone $f(s)$ in $\eta(s)$; this defines a 'choice function' $f : S(d) \to Z(d)$ such that $f(s) \in \eta(s)$. Given the choice function, we can define a set assignment to contour labels $\Psi : L \to \mathbb{P}S(d)$ by

$$\Psi(l) = \begin{cases} \{s \in S(d) : l \in c(f(s))\} & \text{if } l \in L(d) \\ \emptyset & \text{otherwise.} \end{cases}$$

The extension of Ψ to zones and regions satisfies:

1. for any zone $z \in Z(d)$, $\Psi(z) = \{s \in S(d) : f(s) = z\}$, and

2. for any region $r \in R(d)$, $\Psi(r) = \{s \in S(d) : f(s) \in r\}$.

It can be shown that the set assignment $(S(d), \Psi)$ defined above is a model for d. (The proof, which we omit, is similar to that given in [28] for the SD2 system.) $\square$

This ends the material taken from [26]. The existence of a finite, minimal model for each diagram leads to a decision procedure for unitary spider diagrams. The meaning of a compound spider diagram is obtained by using logical connectives $\wedge$ and $\vee$ to take the conjunction and disjunction of the meanings of unitary diagrams, and Howse et al. go on to generalise the unitary decision procedure to the compound case.

Thus, we have examined decision procedures for two monadic first order logics, $\mathcal{MFOL}_=$ and spider diagrams. Each of these logics has the same expressive power, and the process of showing decidability for each system makes use of finite, minimal models.

1.2 Thesis outline

In chapter 2 we present the syntax and semantics of generalized constraint diagrams, making a number of small changes to the original presentation, which are described in section 2.1.1. The changes consist of restrictions to the full system and changes to the semantics, which we are able to simplify. Of the restrictions to the full system, only one is crucial to our exercise: we produce a decidable system from the full, undecidable, notation by excluding universal spiders. We also exclude spiders with several feet. This change is made to shorten and simplify the results and is not essential to the decision procedure, and in chapter 5 we show that our work readily extends to the case of spiders with several feet. The changes to the semantics fall into two categories: simplifications made possible by our restrictions to the system, and simplifications which would be equally applicable to the full system. In the former category, because of the absence of universal spiders, the order in which spiders are 'read' in the semantics is immaterial. Because of this fact, we are able to simplify the original means used to assign semantics to a diagram in a way which makes reasoning about the formulae assigned to diagrams and fragments of diagrams much simpler. The exclusion of spiders with several feet simplifies the semantics in several places; for instance, the habitat of a spider is a single zone, rather than a set of zones. In the category of simplifications to the semantics which would be equally applicable to the full system, we are able to simplify the set of conditions used to test whether an interpretation satisfies a diagram (see section 2.3).

Chapters 3 and 4 contain the main contributions of this thesis, which are the decision procedures for unitary diagrams and for generalized diagrams. In chapter 3, we begin by examining the types of inconsistency found in unitary diagrams and by observing that we cannot, in general, tell whether a unitary diagram is inconsistent by simply examining its arrows (i.e. as is possible with the 'incompatible arrows' of [38], previously discussed on page 17). This observation leads to the class of γ-diagrams, so-called because they extend β-diagrams [28]. A γ-diagram is a unitary diagram in which each zone is shaded, or contains a spider, or both, and in which no arrows are sourced on spiders (see definition 3.1.1). We show that we can determine the consistency of a γ-diagram by inspecting its spiders and arrows. Furthermore, we show that any unitary diagram is equivalent to the disjunction of some set of γ-diagrams and, in section 3.3, develop inference rules that allow us to transform a unitary diagram into what we call its γ components. We base these inference rules on reusable, purely syntactic transformations. Then, to establish the link between the syntactic condition of consistency and the semantic condition of satisfiability, we show that a γ-diagram is satisfiable if and only if it is consistent. Because consistency is a syntactic condition, i.e. one which can be judged solely by an inspection of a diagram's syntax, and because diagrams have finite syntax, this leads to a decision procedure for γ-diagrams. As stated, each unitary diagram is equivalent to its set of γ components, and we use this fact to describe a decision procedure for the unitary fragment of our system.

In chapter 4, we develop a decision procedure for the existential fragment. This works by transforming diagrams through several normal forms, culminating in what we call disjunctive normal form (see section 4.4). The algorithm we define in order to transform a generalized diagram to disjunctive normal form, in combination with the unitary decision procedure, provides a decision procedure for the whole system. If a diagram is in disjunctive normal form, all information in that diagram is contained in a disjunction of leaf nodes. We use this fact to show that, after transforming a diagram into disjunctive normal form, that diagram is satisfiable if and only if one or more of its leaf nodes is satisfiable. That is, we can judge the satisfiability of the diagram by using the unitary decision procedure to test the satisfiability of its leaf nodes.

The transformations and inference rules defined in chapter 4 focus on the tree structure of generalized diagrams rather than the content of unitary diagrams. To transform a generalized diagram, D, into disjunctive normal form, we first 'linearise' $\wedge$-labelled nodes with out-degree of greater than one from D, by reducing their out-degree until they become linear connectives (see section 4.2). In section 4.3, we develop inference rules to 'push' syntax forwards from the root to the leaf nodes. Next we remove all but one of the $\vee$-labelled nodes from the diagram, so that we are left with a diagram which is a disjunction of linear branches (see section 4.4). We then describe the means by which we are able to reduce a linear diagram to a single unitary diagram, whereby the linear diagram is satisfiable if and only if the unitary diagram is satisfiable (see section 4.5).

In the process of reducing the linear diagram to a single unitary diagram, we may encounter inconsistency between two unitary diagrams labelling nodes in the linear diagram. Each of the unitary diagrams in question may be consistent in its own regard but, taken together, they make an inconsistent set of assertions. If this condition exists, we show that the linear diagram is inconsistent and unsatisfiable, enabling us to replace it with $\perp$, the unitary diagram which represents falsity. If the condition does not exist, we are able to remove non-leaf nodes without changing the meaning of the diagram. In this way, we reduce the linear diagram to a semantically equivalent unitary diagram, whose satisfiability can be judged by the application of the unitary decision procedure. Equipped with a decision procedure for linear diagrams, we are able to show that an arbitrary generalized diagram is satisfiable if and only if one or more of the linear branches of its disjunctive normal form is satisfiable (see section 4.6). This completes the decision procedure for the whole system.

In chapter 5, we conclude and discuss potential areas for further work. The discussion focuses on the issue of showing completeness for the system. We show that the strategies used to show completeness of closely related notations are complicated by the fact that generalized constraint diagrams have a second-order semantics. A glossary of terms and notation is supplied as an appendix.

Chapter 2

The syntax and semantics of generalized constraint diagrams

In this chapter we present the syntax and semantics of generalized constraint diagrams, extending the system presented by Stapleton and Delaney [41], building on earlier work on constraint diagrams ([38, 12]) and making use of the notion of equal spiders in [45]. In sections 2.2 and 2.3, on syntax and semantics respectively, we adapt existing work by simplifying its presentation. This simplification is possible because, primarily, the fragment we consider in detail contains less syntax. We will begin by describing constraint diagrams and the underlying notation of Euler diagrams with shading [44], before introducing generalized constraint diagrams.

2.1 Introduction

A drawn generalized constraint diagram is a collection of closed curves called *contours* which represent sets, within an enclosing *boundary rectangle*, representing the universal domain, U. Contours are either *given* or *derived*; given contours represent named sets and are given unique labels, while derived contours represent unnamed subsets of U and are unlabelled. The generalized constraint diagram in Figure 2.1 has three (given) contours, labelled A, B and C, respectively. Containment, intersection and disjointness are represented by the placement of contours, so this diagram asserts $B \cap C = \emptyset$.

A *basic region* is the largest area contained by a contour or the boundary

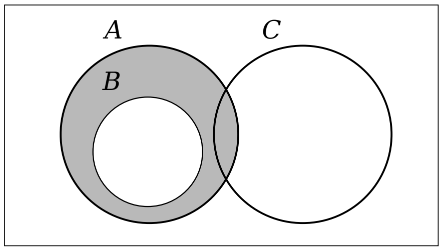


Figure 2.1: Introducing constraint diagrams.

rectangle. A *region* is either a basic region or the intersection, union or difference of two non-empty regions. A *zone* is a region with no other regions inside it. The region outside of all contours is also a zone. The diagram in Figure 2.1 has five zones; one inside A but outside B and C, one inside A and B but outside C, and so on. Shading within a zone places an upper bound on the cardinality of the set represented by that zone; in the absence of other information, a shaded zone represents the empty set. The shading in the diagram in Figure 2.1 tells us that $A - B = \emptyset$ and $A \cap C = \emptyset$. The constraint diagram in Figure 2.1 is also an Euler diagram.

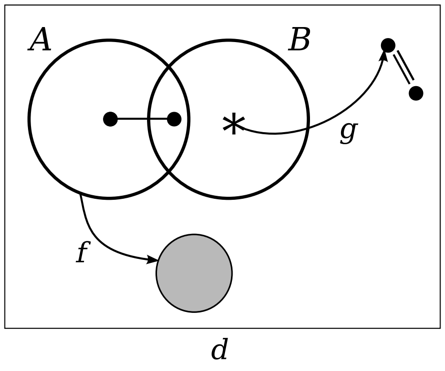


Figure 2.2: The syntax of constraint diagrams.

Constraint diagrams extend the Euler diagram notation described in section 1.1 to include *spiders*, *arrows* and *derived contours*. In drawn diagrams, a spider is a graph whose nodes, called its *feet*, are placed in zones. The region made up of the zones in which a spider's nodes are placed is called its *habitat*. In Figure 2.2, diagram d, there is a spider with two feet whose habitat is the region inside A. The nodes of a spider are either all asterisks or all round dots;

spiders whose nodes are asterisks are called *universal spiders*, while those whose nodes are round dots are called *existential spiders*. In Figure 2.2, diagram d, the spider whose habitat is the region inside A is an existential spider, while the spider whose habitat is the zone inside B but outside A is a universal spider. Semantically, a universal spider represents quantification over all elements in the sets represented by the spider's habitat, which may be empty, while an existential spider asserts the existence of an element in sets represented by the spider's habitat. In this way, existential spiders provide a lower bound for the cardinality of the set represented by their habitat. Since shading provides an upper bound, a shaded zone containing n spiders represents a set which contains exactly n elements. Spiders may be connected by two parallel lines. This device is called a *tie*, and denotes equality. In the diagram in Figure 2.2, there are two spiders in the zone outside of all contours. These spiders are joined by a tie and thus represent a single element.

Arrows are placed in diagrams so that each end of the arrow is directly next to a spider or the edge of a contour. Semantically, arrows represent properties of binary relations and are labelled to indicate the relation in question. The spider or contour at the beginning of the arrow, called the arrow's *source*, represents a restriction on the domain of the relation. The spider or contour at end of the arrow, called its *target*, represents the relational image when the domain is restricted to the set represented by the source, where existential spiders are treated as representing singleton sets. In Figure 2.2, diagram d contains two arrows, labelled f and g, respectively. The arrow labelled f asserts a property of the f relation: when the domain of f is restricted to the set A, the image of f is the empty set.

Figure 2.2 shows a *unitary* diagram. *Compound* diagrams are formed by joining diagrams with logical connectives which state that the semantics of the result is the conjunction or disjunction of its parts. Different varieties of the notation have different ways of indicating that a set of diagrams is joined by a particular connective; Figure 2.3 shows a compound constraint diagram in Stapleton's notation [38]. The meaning of this diagram is given by the conjunction of the meanings of the two unitary diagrams it contains.

As discussed in section 1.1.1, a diagrammatic notation that represents quantifiers must provide an unambiguous way of combining those quantifiers. In

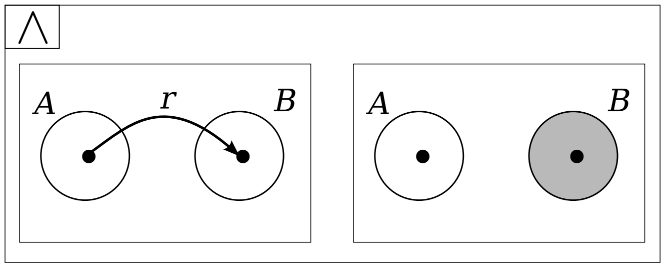


Figure 2.3: A compound constraint diagram.

that section we discussed augmented constraint diagrams which include an extra-diagrammatic reading tree. The system of *generalized* constraint diagrams [41] solves the same problem by allowing diagrammatic elements to be introduced piece-wise, in what may be thought of as a movie strip or timeline. Figure 2.4 shows an example.

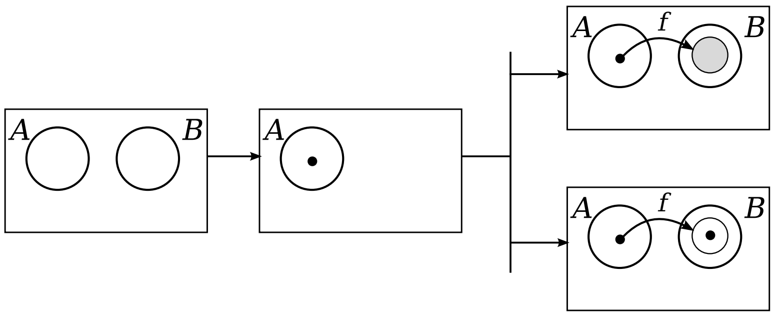


Figure 2.4: A generalized constraint diagram.

A GCD is a tree structure of unitary diagrams which imposes a partial order on the syntactic elements of the unitary diagrams. The root node of the tree is labelled by a unitary diagram, called the *root diagram*. Each node below the root is labelled by either a unitary diagram or a connective. The connectives used are $\wedge$ and $\vee$, represented in drawn diagrams by a fork and a vertical bar, respectively. Each diagram below the root may add some new piece of syntax to the diagrams preceding it. Indeed, it may also remove syntax. Constraints are placed on what may be added to ensure no ambiguity arises.

The structure of a GCD provides the reading order in which its semantics is interpreted. In Figure 2.5, d_2 can be thought of as 'in the scope of' d_1. Because the universal spider is introduced in d_1, before the arrow and existential spider are introduced in d_2, the informal meaning of the diagram is that “every man is the son of exactly one woman,” not “there is a woman of whom every man is the son”.

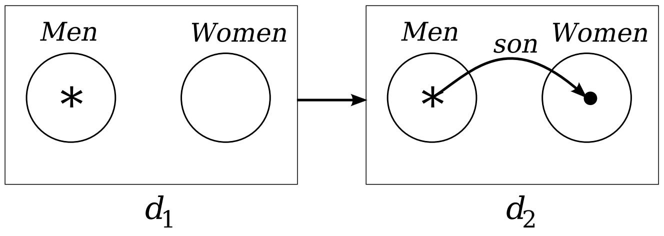


Figure 2.5: Reading order in generalized constraint diagrams.

2.1.1 Restrictions imposed on generalized constraint diagrams in our system

There are several variations of the constraint diagram notation, each devised to have specific properties. The system we consider is produced by applying certain restrictions to generalized constraint diagrams. The restrictions we place on GCDs are that diagrams may not contain universal spiders, and that spiders have feet placed in a single zone. Additionally, we add the syntactic device of ties between spiders [45], representing their equality. Thus, spiders' feet may be joined by ties but may not represent disjunctive information. This means that unitary diagrams in our system are α-diagrams [22], in which all spiders have a single foot, and we call the resulting system the *existential fragment* of generalized constraint diagrams. We will refer to existential spiders as ‘spiders’ from here on unless it is not clear from the context which type of spider we mean.

2.2 Syntax

We will now introduce the syntax of GCDs, adapting the definitions in [41] with the restrictions discussed above, and beginning with the case of unitary diagrams.

2.2.1 Generalized unitary constraint diagrams

The contours of a unitary diagram are provided by the sets $\mathcal{GC}$, of given contours, and $\mathcal{DC}$, of derived contours. In the abstract syntax we identify contours with their labels, so the contents of $\mathcal{GC}$ are what is used to label contours in the drawn diagram. Derived contours have no label, and we consider the contents of $\mathcal{DC}$ to correspond directly to the derived contours of the concrete diagram. The set $\mathcal{AL}$ contains labels for arrows and $\mathcal{S}$ contains spiders. We assume that these four sets are pairwise disjoint. Since we wish never to run out of spiders or derived contours we further assume the sets $\mathcal{S}$ and $\mathcal{DC}$ are countably infinite. As with Euler diagrams, a zone is a pair of disjoint sets of contour labels and the set of all zones, $\mathcal{Z}$, has the following type:

$$\mathcal{Z} : \mathbb{P}(\mathcal{GC} \cup \mathcal{DC}) \times \mathbb{P}(\mathcal{GC} \cup \mathcal{DC}).$$

A *region* is any collection of zones. Syntactically, arrows are composed of a *label*, *source* and *target*. Spiders and contours, both given and derived, may act as sources and targets of arrows. Figure 2.6 illustrates the possible sources and targets for arrows. The definitions in the remainder of this subsection extend those in [42] and [45].

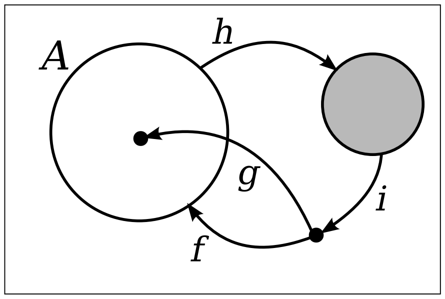


Figure 2.6: Sources and targets of arrows.

Definition 2.2.1. An **arrow end** is either a contour or a spider. An **arrow** is an ordered triple (l, s, t) where $l \in \mathcal{AL}$, s is an arrow end called the **source** and t is an arrow end called the **target**.

Two spiders in the same zone can be joined by a tie to assert equality, indicating that they represent the same element. This aspect of the syntax is not part of the original presentation of generalized constraint diagrams and adapts work in [45]. At the abstract level we provide each unitary diagram, d, with an equality relation on $S(d)$ called τ_d, where $(x, y) \in \tau_d$ if and only if the spiders x and y are joined by a tie. Certain ties which may be implied are omitted from drawn diagrams to avoid clutter. For any spider $x \in S(d)$, we have $(x, x) \in \tau_d$ and so each spider is considered to be tied to itself; these ties are omitted from drawn diagrams. Furthermore, for all $x, y, z \in S(d)$, the equality relation is symmetric, so that if $(x, y) \in \tau_d$ then $(y, x) \in \tau_d$, and transitive, so that if $(x, y) \in \tau_d$ and $(y, z) \in \tau_d$, then $(x, z) \in \tau_d$. Thus, τ is an equivalence relation which partitions the set of spiders. In drawn diagrams the ties implied by the transitive property of τ may be omitted. That is, if spiders y and z are joined by a tie and we add a tie between a third spider, x, and y, the tie between x and z can be omitted from the drawing but is present in the abstract syntax.

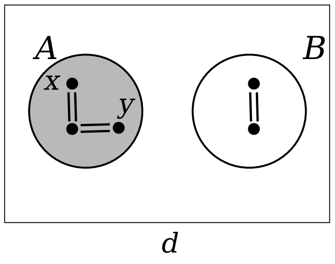


Figure 2.7: Illustrating ties between spiders.

In Figure 2.7, there are two spiders in the zone $(\{B\}, \{A\})$. Because these spiders are joined by a tie, they must represent the same element in any interpretation that satisfies d. Thus, these two spiders indicate that there is at least one element in the set represented by B. There are three spiders joined by ties in the shaded zone $(\{A\}, \{B\})$, indicating that the represented set contains exactly one element. The spider labelled x is not tied directly to the spider labelled y, but we know that $(x, y) \in \tau_d$ by the transitive property of τ, and it greatly reduces clutter in diagrams to omit such ties.

Definition 2.2.2. A **generalized unitary diagram** is a tuple

$$d = (C, Z, Z^*, S, \eta, \tau, A)$$

which satisfies the following:

1. $C = C(d)$ is a finite set of contours, that is $C \subset \mathcal{GC} \cup \mathcal{DC}$.
2. $Z = Z(d)$ is a finite set of zones such that:
   (a) for each zone $(in, out) \in Z(d)$, $in \cup out = C(d)$ and $in \cap out = \emptyset$,
   (b) for each contour $c \in C(d)$, there is a zone $(in, out) \in Z(d)$ where $c \in in$, and
   (c) the zone outside of all contours, $(\emptyset, C(d))$, is in $Z(d)$.
3. $Z^* = Z^*(d)$ is a set of shaded zones such that $Z^*(d) \subseteq Z(d)$.
4. $S = S(d)$ is a finite set of spiders.
5. $\eta = \eta_d$ is function $\eta_d : S(d) \to Z(d)$ which returns the habitat of every spider.
6. $\tau = \tau_d$ is a binary relation on $S(d)$ which satisfies the following: for all $x, y, z \in S(d)$,
   (a) $(x, x) \in \tau_d$,
   (b) $(x, y) \in \tau_d \Rightarrow (y, x) \in \tau_d$,
   (c) $((x, y) \in \tau_d \wedge (y, z) \in \tau_d) \Rightarrow (x, z) \in \tau_d$.
7. $A = A(d)$ is a finite set of arrows such that for each arrow $(l, s, t) \in A(d)$, s and t are each either a contour in $C(d)$ or a spider in $S(d)$.

Additionally, the diagram $\perp = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ is a generalized unitary constraint diagram. The set of all generalized unitary diagrams is denoted $\mathcal{UD}$.

As we have said, the spiders inhabiting a zone provide a lower bound on the cardinality of the set that zone represents. Note that, in general, the lower bound is provided not by the number of spiders, but by the number of equivalence classes under τ.

Definition 2.2.3. A spider **inhabits** a zone in a unitary diagram d if it is placed in that zone in d. The zone inhabited by a spider is also called its **habitat**. Let r be a subset of $Z(d)$. We define the set of spiders inhabiting the zones in r in d, denoted $S(r, d)$, to be

$$S(r, d) = \{s \in S(d) : \eta_d(s) \in r\}$$

We next define notation which allows us to refer to the sets of zones which are inside a particular contour, and the set of spiders inhabiting those zones. For example, in Figure 2.8, the set of zones containing A is

$$\{(\{A\},\{B,C\}),(\{A,B\},\{C\}),(\{A,C\},\{B\}),(\{A,B,C\},\emptyset)\}.$$

There are three spiders inhabiting a zone in this region, two of which are joined by a tie.

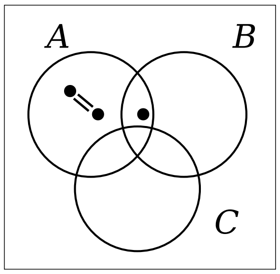


Figure 2.8: Zones containing a contour and the spiders inhabiting those zones.

Definition 2.2.4. Let d be a generalized unitary diagram and c a contour label in $C(d)$. We define $Z(c,d)$, the zones in d which include c, to be

$$Z(c,d)=\{(in,out)\in Z(d):c\in in\}.$$

The set of spiders **inhabiting** c in diagram d, denoted $S(c,d)$, is defined to be

$$S(c,d)=S(Z(c,d),d).$$

For convenience in later definitions we extend S to individual spiders such that for each spider x in $S(d)$, $S(x,d)=\{x\}$.

Definition 2.2.5. Let d be a generalized unitary diagram. We define the **derived contours** of d, denoted $DC(d)$, as follows:

$$DC(d)=\{c\in C(d)\cap\mathcal{DC}\}.$$

If a unitary diagram, d, contains all possible zones, then the diagram obtained by removing all spiders and arrows from d is a Venn diagram. We define the set of all possible zones for d, some of which may be missing, which we call the *Venn zone set* of d, and the set of zones which are missing from d.

Definition 2.2.6. Let d be a generalized unitary diagram. We define the **Venn zone set** of d, denoted $VZ(d)$, as the set $\{(a, C(d) - a) : a \in \mathbb{P}C(d)\}$. We define the **missing** zones of d, denoted $MZ(d)$, as those in the set $VZ(d) - Z(d)$.

2.2.2 Generalized constraint diagrams

A generalized constraint diagram is a bipartite tree in which each node is labelled with either a generalized unitary diagram or one of the connectors $\wedge$ and $\vee$. At the drawn level, connectors are represented by forks and vertical bars, respectively; each fork or vertical bar may have more than two branches, or even only one branch. The tree is directed, and for each diagram-labelled node the information provided by its ancestors is available. The diagram in Figure 2.9 has three nodes. The root node is labelled by the unitary diagram d_1. The immediate descendant of the root node is an $\wedge$-labelled node which is not shown in the diagram. When it is relevant to the discussion, such nodes will sometimes be depicted in meta-diagrams but, to avoid clutter, $\wedge$-labelled nodes with out-degree of one are not depicted in generalized diagrams. Finally, the diagram in Figure 2.9 has a leaf node labelled by the unitary diagram d_2. The semantic formula for the diagram is derived from the conjunction of the meanings of d_1 and d_2, but d_2 is inside the scope of d_1 and not vice versa. That is, as diagram-labelled nodes are added to a graph their meaning is interpreted in the context of their ancestor nodes. In Figure 2.9 then, we read the information provided by d_2 in combination with that provided by its ancestor node and are able to conclude, informally, that A has exactly one element and that element is related to an element of $U - A$ under f.

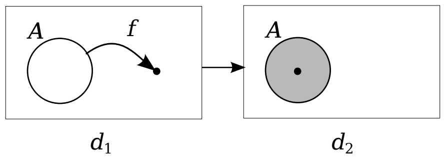


Figure 2.9: Generalized diagrams and scope.

Diagram-labelled nodes may introduce diagrammatic elements such as spiders and arrows; the full system provides rules to ensure that universal spiders are introduced in such a way that the scope of the quantifiers they represent is clear [41]. Our system does not include universal spiders and, since it does not matter in which order existential quantifiers are read, these rules are not needed.

Figure 2.10 is a meta-diagram which illustrates the structure of a generalized diagram. We will use this type of meta-diagram frequently throughout the work, whenever the discussion is mostly or entirely concerned with the structure of diagrams and it is helpful to ignore the unitary diagrams used as labels. In this type of meta-diagram, diagram-labelled nodes are represented by solid rectangles, and solid circles represent connective-labelled nodes. Each node is given a name, n_1, n_2, etc., and the label of a node (i.e. a logical connective or the name of a unitary diagram) may be shown above the node, as is done in Figure 2.10.

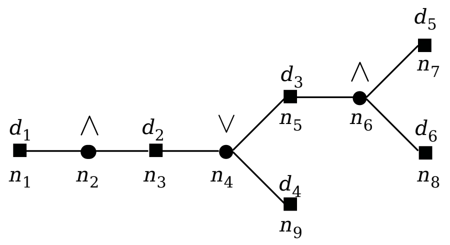


Figure 2.10: The structure of a generalized diagram.

In Figure 2.10, nodes are named n_1 to n_9. Above each node is shown its label: unitary diagrams d_1 to d_6, which are assumed to be defined, and the connectors $\vee$ and $\wedge$. In Figure 2.11 that structure is visualised. The root node, n_1, is labelled by a unitary diagram, d_1. The immediate descendant of the root node is a connective-labelled node, n_2. The node n_2 is shown in Figure 2.10, but is missing from Figure 2.11. Connective-labelled nodes with an out-degree of one are suppressed in drawn diagrams to reduce clutter, but are present in the abstract representation of the diagram. The next node, n_3, is labelled by a diagram, d_2. Because d_1 is joined to d_2 with a straight line in Figure 2.11, we can informally consider them to be in conjunction except that, semantically, the quantifier introduced by d_2 (represented by the spider) is not in the scope of d_1. Since, unlike the original presentation of GCDs in [41], our existential fragment does not include universal spiders, the order in which d_1 and d_2 are read could, in fact, be reversed without changing the meaning of the diagram. Unitary diagrams cannot, however, be moved in arbitrary ways without a change in meaning; for example, exchanging d_2 and d_3 in 2.11 results in a diagram with a different meaning.

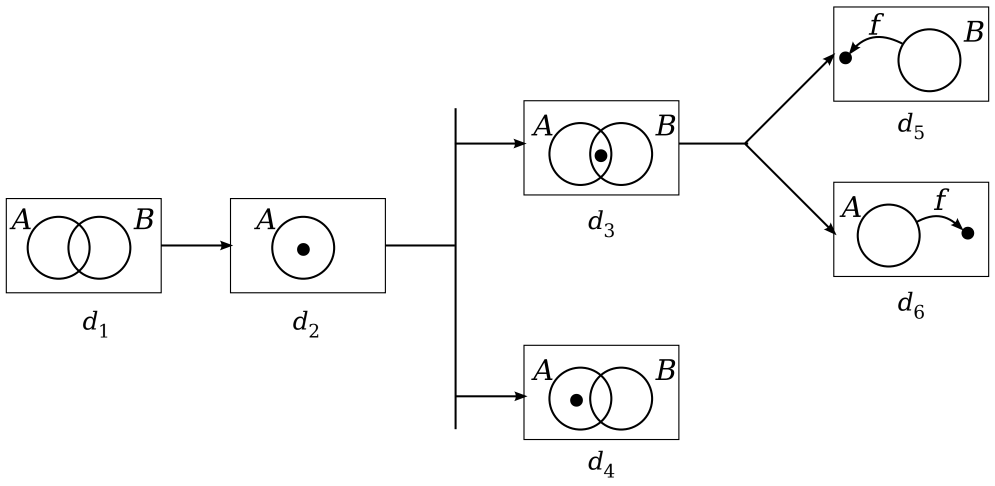


Figure 2.11: Visualising the structure of a generalized diagram.

After the node labelled by the unitary diagram d_2, (node n_3, Figure 2.10), the next node, n_4, is labelled by an $\vee$ connector. Unlike the node n_2, n_4 is shown in Figure 2.11, and is represented by a vertical bar. The disjunction of the meanings of the two branches of the vertical bar is added to the meaning of the diagram as a whole. Diagram d_4 labels a leaf node. Attached to d_3 is an $\wedge$ connector, with the effect that the meaning of $d_5 \wedge d_6$ is added to the meaning of d_3. Informally, if we take the labels d_1, d_2, etc. to stand for the meaning of the unitary diagrams in question, the meaning of the diagram as a whole is given by the formula:

$$d_1 \wedge d_2 \wedge ((d_3 \wedge (d_5 \wedge d_6)) \vee d_4).$$

We will now formalise the structure of generalized diagrams, extending the definition given in [41]. Nodes, or vertices, are drawn from a countably infinite set $\mathcal{V}$.

Definition 2.2.7. A **generalized constraint diagram** is a (node) labelled, directed bipartite tree $D = (V, W, E, l)$ which satisfies the following:

1. V and W are disjoint sets of nodes, and $V \neq \emptyset$.

2. $E : (V \times W) \cup (W \times V)$ is a set of edges which satisfies the following:

   (a) There is one root node in E, which is in the set V.

   (b) Every node in V except the root node has an in-degree of 1.

   (c) Every node in V has an out-degree of at most 1.

   (d) Every node in W has an in-degree of 1 and an out-degree which is greater than 0.

3. $l : (V \cup W) \rightarrow \mathcal{UD} \cup \{\wedge, \vee\}$ is a labelling function such that:

   (a) The nodes in V are labelled by generalized unitary constraint diagrams.

   (b) The nodes in W are labelled by either $\wedge$ or $\vee$.

In order to describe the tree structure of individual generalized diagrams without listing sets of vertices and edges, etc., we use a shorthand notation in which the symbols $\wedge$ and $\vee$ mean the same as the fork and vertical bar found in drawn diagrams. When a connective-labelled node has more than one immediate descendant, the connective, $\wedge$ or $\vee$, is used as a prefix operator and followed by a collection of nodes in set notation. For example, if the abstractions of the unitary diagrams d_1, d_2 etc are defined, then the diagram in Figure 2.11 has abstract syntax described by

$$d_1 \wedge d_2 \wedge (\vee\{d_3 \wedge (\wedge\{d_5, d_6\}), d_4\}).$$

Our convention is to use lower case letters d, $d_1, \ldots, d_n$ and so on to denote generalized unitary diagrams and upper case letters D, $D_1, \ldots, D_n$, etc., to denote non-unitary generalized diagrams. Given a generalized diagram D, we will often need to refer to the subsets of the nodes, edges and labels of D which are themselves generalized diagrams. We call such diagrams *sub-diagrams* of D.

Definition 2.2.8. Let $D_1 = (V_1, W_1, E_1, l_1)$ and $D_2 = (V_2, W_2, E_2, l_2)$ be generalized diagrams. We say that D_1 is a **sub-diagram** of D_2, denoted $D_1 \subseteq D_2$ if and only if the following is true:

1. $V_1 \subseteq V_2$,

2. $W_1 \subseteq W_2$,

3. $E_1 = E_2 \cap ((V_1 \times W_1) \cup (W_1 \times V_1))$, and

4. $l_1 = l_2|_{V_1 \cup W_1}$.

As an example of sub-diagrams, consider Figure 2.12, diagram D_1. Each connected set of nodes from the underlying graph structure of D_1, which is also a generalized diagram, is a sub-diagram of D_1. Figure 2.12 shows the following sub-diagrams of D_1:

1. d_1,

2. $d_1 \wedge d_2$,

3. $d_1 \wedge \{d_2, d_4\}$.

There are other sub-diagrams of D_1 which are not shown in Figure 2.12. However, $\wedge\{d_1 \wedge d_2, d_3\}$ is not a sub-diagram of D_1 since, as the root node is not labelled by a diagram, it is not a generalized diagram. The diagram $d_1 \wedge d_3$ is not a sub-diagram of D_1 since it contains an edge which is not in D_1.

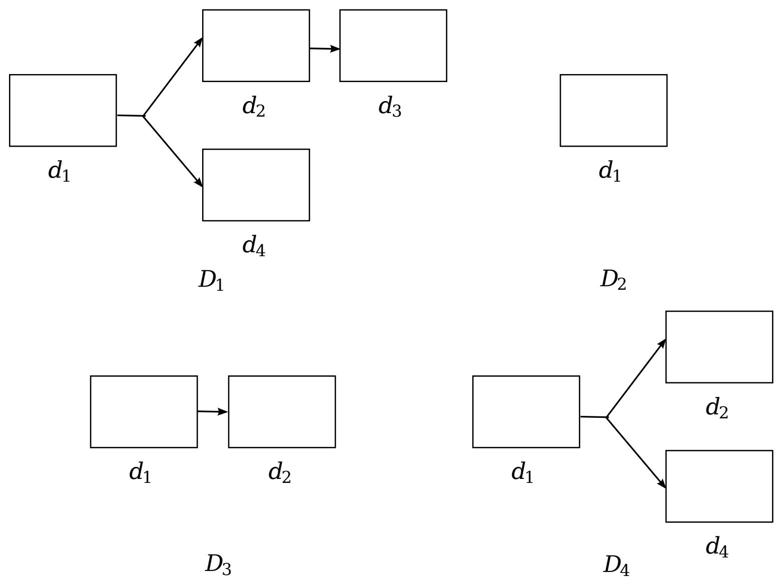


Figure 2.12: Diagrams and sub-diagrams.

Figure 2.13 shows a meta-diagram in which connective-labelled nodes are labelled n_2 and n_4. Node n_2 is labelled by $\wedge$ and has an out-degree of 1. We say

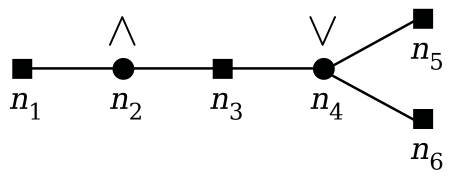


Figure 2.13: Linear and non-linear connectives.

that n_2 is a *linear* connective, Node n_4 has an out-degree of 2; we say that n_4 is a *non-linear* connective.

Definition 2.2.9. Let D be a generalized diagram which includes a connective-labelled node n. We say that n is a **non-linear** connective of D if and only if n has an out-degree of greater than one: $|\{(n, n') \in E\}| > 1$. Otherwise, we say that n is a **linear** connective. If D contains only linear connectives we say that D is a linear generalized diagram.

We define a function which supplies the root of a generalized diagram.

Definition 2.2.10. Let D be a generalized diagram. Then we define the function *root*, so that *root*(D) returns the root vertex of D.

The edges of a diagram give rise to a partial order on its nodes which we characterise as the sets of *ancestors* and *descendants* of a given node. Figure 2.14 is a meta-diagram in which nodes are labelled with V or W to show which set the node belongs to. The immediate ancestor of n_6 is n_5, which is another way of saying that the edge (n_5, n_6) is in the diagram. The vertex n_6 has two immediate descendants, n_7 and n_8.

Note that immediate ancestors are unique but immediate descendants are not, in general. Also, the sets of all ancestors and descendants of a node are given by the transitive closures of the immediate ancestor and immediate descendant relations respectively.

Definition 2.2.11. Let $D = (V, W, E, l)$ be a generalized diagram and let n_1 and n_2 be two nodes in $V \cup W$. We say that n_1 is the **immediate ancestor** of n_2 and n_2 is an **immediate descendant** of n_1 if and only if the edge (n_1, n_2) is in E. We denote the set of immediate descendants of n_1 in D as *ImmDes*(n_1, D). We denote the set of all ancestors of n_2, which is the transitive closure of the

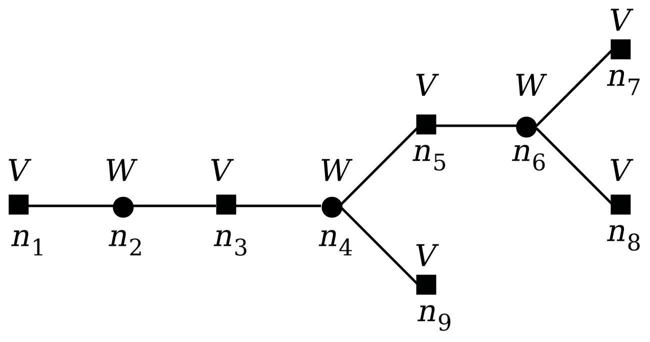


Figure 2.14: The ancestors and descendants of a node.

immediate ancestor relation, as $Anc(n_2, D)$, and the set of all **descendants** of n_1 in D, which is the transitive closure of the immediate descendant relation, as $Des(n_1, D)$.

We will also need to refer to the nearest ancestor or descendant which is labelled by a diagram. We call these the *immediate diagram-labelled* ancestors and descendants. As with (diagram- or connective-labelled) ancestors and descendants, immediate diagram-labelled ancestors are unique but immediate diagram-labelled descendants need not be. In Figure 2.14, n_7 and n_8 are necessarily labelled by unitary diagrams since they are leaf nodes. It follows that n_6 must be labelled by a connective and that the immediate diagram-labelled ancestor of both n_7 and n_8 is node n_5.

Definition 2.2.12. Let D be a generalized diagram which contains diagram-labelled nodes n_1 and n_2. We say that n_1 is the **immediate diagram-labelled ancestor** of n_2 if and only if there is a path of length two from n_1 to n_2. If this is the case, we say that n_2 is an **immediate diagram-labelled descendant** of n_1.

2.3 Semantics

The semantics we present is taken from Stapleton and Delaney [41] with adaptations that reflect the fact that we work in the existential fragment of the system. We are also able to simplify the semantics in several places. This section is therefore an adaptation of existing work.

The first stage of the process of assigning semantics to generalized diagrams is to interpret the given contour labels as subsets of some specified universal set U, and the arrow labels as binary relations on U, giving an *interpretation*. We call the mapping for contour labels Ψ, and the mapping for arrow labels Φ. For example, an interpretation I given by

1. $U = \{1, 2\}$,
2. $\Psi = \{(A, \{1\}), (B, \{2\})\}$,
3. $\Phi = \{(f, \{(1, 2)\})\}$,

satisfies the diagram in Figure 2.15. We mean by this that the diagram can be interpreted via this universe and mappings without contradictions arising (the notion of satisfaction is formalised later in this section). The interpretation J, given by

1. $U = \{1, 2\}$,
2. $\Psi = \{(A, \{1, 2\}), (B, \emptyset)\}$,
3. $\Phi = \{(f, \{(1, 2)\})\}$,

does not satisfy the diagram, since J contradicts the assertions of the diagram that B contains at least one element and that an element of A is related to all elements of B under f; either one of these properties is sufficient for J not to satisfy the diagram.

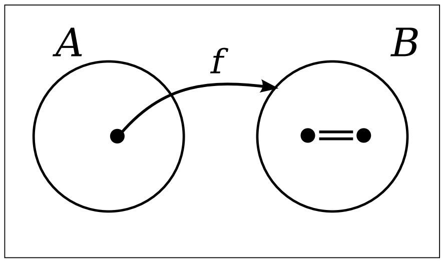


Figure 2.15: Interpretations of a unitary diagram.

By converting the diagram to a sentence in symbolic logic using the process described in this section, it can be shown that I satisfies the diagram but J

does not. These symbolic sentences are used to determine whether an interpretation satisfies a diagram. The sentence assigned to a given diagram allows us to determine whether an interpretation satisfies that diagram.

Definition 2.3.1. An **interpretation** is a triple, (U, Ψ, Φ), where U is a non-empty set, $\Psi : \mathcal{GC} \to \mathbb{P}U$ is a function that maps given contours to subsets of U and $\Phi : \mathcal{AL} \to \mathbb{P}(U \times U)$ maps arrow labels to binary relations on U [43].

It is often essential and sometimes merely convenient to extend the mapping Ψ to interpret other diagrammatic elements: derived contours, zones and individual spiders. We therefore define *extended interpretations.*

Definition 2.3.2. Let $I = (U, \Psi, \Phi)$ be an interpretation. We define the **extended interpretation** $I' = (U, \Psi', \Phi)$ where Ψ' extends Ψ,

$$\Psi' \colon \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S} \cup \mathcal{Z} \to \mathbb{P}U,$$

and satisfies the following.

1. For each zone, $(in, out) \in \mathcal{Z}$, 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

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

3. For each spider, x, $|\Psi'(x)| = 1$ and $\Psi'(x) \subseteq U$.

The definition of Ψ' for zones has a useful corollary.

Corollary 2.3.1. Let (in, out) be a zone, let c be a contour label not in $in \cup out$ and let $m = (U, \Psi', \Phi)$ be an extended interpretation which interprets c. Then

$$\Psi'(in, out) = \Psi'(in \cup \{c\}, out) \cup \Psi'(in, out \cup \{c\}).$$

For unitary diagrams, d, we will specify a collection of conditions 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. For an interpretation to satisfy a diagram, d, these conditions must hold with regard to the interpretation and the semantic formula assigned to d.

The first condition, the *plane tiling condition*, states that the union of the sets represented by the zones in d is the universal set. The *shaded zones condition* ensures that shading imposes an upper bound on the cardinality of the sets represented by shaded zones. The *spiders habitat condition* asserts that each universal element represented by a spider is a member of the set represented by the zone the spider inhabits. The *spiders' distinctness condition* states that spiders not joined by a tie represent different universal elements, while any two spiders joined by tie represent the same element. The *arrows condition* states that, for each arrow, the set represented by the target is indeed the image of the relation represented by the arrow label, when the domain is restricted to the set represented by the arrow's source. After formalising the conditions we present a concrete example.

Definition 2.3.3. Let U be a set, let R be a binary relation on U and let A be a subset of U. We define the **image** of R when the domain is restricted to A, denoted $A.R$, as follows:

$$A.R = \{y \in U : \exists x \in A\,((x, y) \in R)\}.$$

Definition 2.3.4. Let d ($\neq \perp$) be a generalized unitary diagram and let (U, Ψ, Φ) be an interpretation with extension (U, Ψ', Φ).

1. The **plane tiling condition** for d, denoted $PTC(d)$, asserts that the union of the sets represented by the zones is the universal set:

$$\Psi'(Z(d)) = U.$$

2. The **shaded zones condition** for d, denoted $SZC(d)$, asserts that all of the elements in the sets represented by shaded zones are represented by

spiders:
$$\bigwedge_{z \in Z^*(d)} \Psi'(z) = \bigcup_{x \in S(z,d)} \Psi'(x).$$

3. The **spiders' habitats condition** for d, denoted $SHC(d)$, asserts that the elements represented by the spiders are in the sets represented by their habitats:
$$\bigwedge_{x \in S(d)} \Psi'(x) \subseteq \Psi'(\eta_d(x)).$$

4. The **spiders distinctness condition** for d, denoted $SDC(d)$, asserts that two spiders represent the same element if and only if they are joined by a tie:
$$\bigwedge_{x,y \in S(d) \wedge x \neq y} \Psi'(x) = \Psi'(y) \Leftrightarrow (x,y) \in \tau_d.$$

5. The **arrows condition** for d, denoted $AC(d)$, 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:
$$\bigwedge_{(l,s,t) \in A(d)} \Psi'(s).\Phi(l) = \Psi'(t).$$

The conjunction of the above five conditions is called the **formula for *d***, denoted $form(d)$.

Note that the arrows condition is a simplification of that in the original presentation. The simplification is made possible by extending the function $S : \mathcal{Z} \cup \mathcal{C} \cup \mathcal{UD} \to \mathbb{P}\mathcal{S}$ to operate on individual spiders. Thus, given a unitary diagram d and a spider $x \in S(d)$, $S(x,d) = \{x\}$. The result of this is that the arrows condition is shortened from four cases (the potential types of source and target of arrows) to one.

An equivalent way of stating the plane tiling condition is by the *containment condition* [38, p54].

Definition 2.3.5. Let d ($\neq \perp$) be a generalized unitary diagram and let (U, Ψ', Φ) be an extended interpretation. The set represented by each basic region is the same as that represented by its containing contour:

$$\bigwedge_{c \in C(d)} \Psi'(c) = \bigcup_{(in,out) \in Z(d) \wedge c \in in} \Psi'(in, out).$$

Lemma 2.3.1. The plane tiling condition and the containment condition are equivalent.

Proof. See Stapleton [38, p54] for a proof of the equivalence of the plane tiling and containment conditions for spider diagrams, which is also valid for generalized unitary diagrams. □

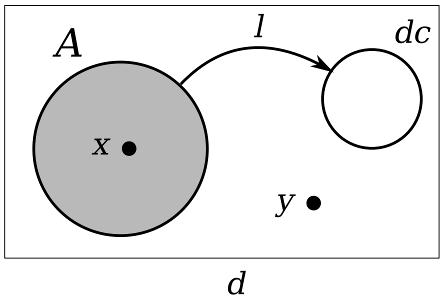


Figure 2.16: Illustrating GCD semantics.

To illustrate the process of assigning a formula to a unitary diagram, consider diagram d in Figure 2.16, where the derived contour dc is given a meta-level label for convenience, as are the spiders. This diagram has the formula $form(d)$, given by

$$form(d) = PTC(d) \wedge SZC(d) \wedge SHC(d) \wedge SDC(d) \wedge AC(d).$$

The five conditions can be given in full as follows.

1. The plane tiling condition, $PTC(d)$:

$$\Psi(\{A\}, \{dc\}) \cup \Psi(\{dc\}, \{A\}) \cup \Psi(\emptyset, \{A, dc\}) = U.$$

2. The shaded zones condition, $SZC(d)$:

$$\Psi(\{A\}, \{dc\}) = \bigcup_{x \in S((\{A\},\{dc\}),d)} \Psi(x).$$

3. The spiders' habitats condition, $SHC(d)$:

$$\Psi(x) \subseteq \Psi(\{A\}, \{dc\}) \wedge \Psi(y) \subseteq \Psi(\emptyset, \{A, dc\}).$$

4. The spiders distinctness condition, $SDC(d)$: $\Psi(x) \neq \Psi(y)$.

5. The arrows condition, $AC(d)$: $\Psi(A).\Phi(l) = \Psi(dc)$.

In the original presentation [41] of the semantics of generalized constraint diagrams, spiders and derived contours each give rise to quantification expressions in the sentence assigned to a diagram. Separate quantification expressions are produced for derived contours. We have simplified this presentation by removing the need for quantification expressions relating to spiders. Instead, in a unitary diagram the mapping introduced to interpret derived contours and zones is extended to spiders:

$$\exists \Psi' : \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S} \cup \mathcal{Z} \rightarrow \mathbb{P}U.$$

Our differing approach means that, instead of referring to spiders via variables which refer to universal set elements, we apply our extended mapping, Ψ, to syntax-level spiders, i.e. elements of $\mathcal{S}$. This choice simplifies details of the arrows condition. Also, instead of producing a new existential quantifier to prefix the formula for each unitary diagram in an extended interpretation, we are able to make use of a single quantifier that asserts the existence of an extension, which appears at the beginning of the formula.

We generalise the functions C and S from the unitary to the compound case.

Definition 2.3.6. Let $D = (V, W, E, l)$ be a generalized diagram. Then we define the **contour set** of D, denoted $C(D)$, as follows:

$$C(D) = \bigcup_{n \in V} C(l(n)).$$

Similarly, the **spider set** of D, denoted $S(D)$, is defined as follows:

$$S(D) = \bigcup_{n \in V} S(l(n)).$$

We can now describe the process of constructing a formula for a generalized constraint diagram, allowing us to determine whether a given interpretation agrees with the meaning of the diagram. We begin by producing a formula for the root and build up an expression which yields a sentence for the entire generalized diagram.

Definition 2.3.7. Let D be a generalized constraint diagram and let n_i be a node in D labelled by the unitary diagram d_i. Let $IDL(n, D)$ be the function which returns the immediate diagram-labelled descendants of the node n in D, and let (U, Ψ', Φ) be an extended interpretation. We define the **formula** for n_i in the context of D, denoted $SemForm(n_i, D)$, as follows.

1. If n_i is a leaf node then $SemForm(n_i, D)$ is the formula $form(d_i)$.
2. If n_i is not a leaf node of D, and the immediate descendant of n_i is labelled by $\vee$, then $SemForm(n_i, D)$ is the formula
$$form(d_i) \wedge (\bigvee_{n_j \in IDL(n_i,D)} SemForm(n_j, D)).$$
3. Otherwise n_i is not a leaf of D and the immediate descendant of n_i is labelled by $\wedge$, in which case $SemForm(n_i, D)$ is the formula
$$form(d_i) \wedge \bigwedge_{n_j \in IDL(n_i,D)} SemForm(n_j, D).$$

The formula for D, denoted $form(D)$, is $SemForm(root(D), D)$.

We are now able to formally define what it means for an interpretation to satisfy a diagram.

Definition 2.3.8. Let D be a generalized diagram with root node n and let $I = (U, \Psi, \Phi)$ be an interpretation. We say that I **satisfies** D if there exists an extension of I, I', under which $form(D)$ is true. If I satisfies a generalized diagram D then we say that I is a **model** for D. Let Ψ' be an extension of Ψ with the following signature:

$$\Psi' : \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S} \cup \mathcal{Z} \rightarrow \mathbb{P}U.$$

If $form(D)$ is true under Ψ' then Ψ' is called a **valid** extension for D.

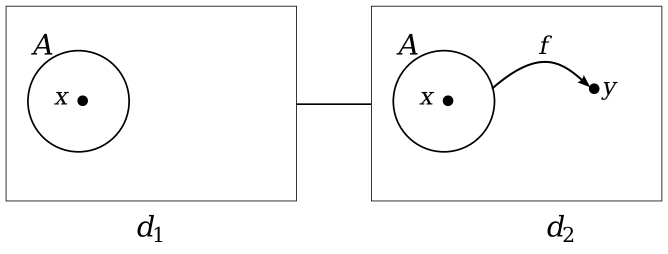


Figure 2.17: Interpreting a GCD.

For any generalized constraint diagram, the formula of the root node is a sentence. For example, given an extended interpretation $I' = (U, \Psi', \Phi)$, the diagram with two nodes in Figure 2.17 has the formula $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$, which we can give in full as follows. First, the conditions for d_1, where $\top$ denotes truth:

1. $PTC(d_1) = (\Psi'(\{A\}, \emptyset) \cup \Psi'(\emptyset, \{A\}) = U)$,
2. $SZC(d_1) = \top$,
3. $SHC(d_1) = \Psi(x) \subseteq \Psi(\{A\}, \emptyset)$,
4. $SDC(d_1) = \top$,
5. $AC(d_1) = \top$.

Next, the conditions for d_2:

1. $PTC(d_2) = (\Psi'(\{A\}, \emptyset) \cup \Psi'(\emptyset, \{A\}) = U)$,
2. $SZC(d_2) = \top$,
3. $SHC(d_2) = \Psi(x) \subseteq \Psi(\{A\}, \emptyset) \ \wedge \ \Psi(y) \subseteq \Psi(\emptyset, \{A\})$,
4. $SDC(d_2) = (\Psi(x) \neq \Psi(y))$,
5. $AC(d_2) = (\Psi(A).\Phi(f) = \Psi(y))$.

Finally, the formula for the diagram as a whole, omitting conditions which are trivially true:

$$\exists \Psi' : \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S} \rightarrow \mathbb{P}U$$
$$\Big(PTC(d_1) \wedge SHC(d_1) \wedge (PTC(d_2) \wedge SHC(d_2) \wedge SDC(d_2) \wedge AC(d_2))\Big).$$

Thus, we have defined the syntax of generalized constraint diagrams and identified the differences between our fragment and the original presentation. We have also defined the semantics of generalized diagrams, including the notion of satisfiability, and given a number of examples that show how the meaning of a generalized diagram is constructed.

Chapter 3

A decision procedure for the unitary existential fragment

Our goal in this chapter is to develop a decision procedure which judges the satisfiability of a unitary diagram. In order to do this, we need to identify the syntactic properties which distinguish those diagrams which are satisfiable. Diagrams with these properties are *consistent*, and those without them are *inconsistent*. In this chapter we define the syntactic property of consistency and show that it corresponds exactly to the semantic property of satisfiability; that is, a unitary diagram is satisfiable if and only if it is consistent. The conditions under which we can do this depend on the syntactic properties of the class of diagrams called γ-diagrams, defined in section 3.1, and on their use a reduction class. That is, we define consistency for the class of γ-diagrams, and show that any unitary diagram is equivalent to the disjunction of a particular set of γ-diagrams, called its γ components. By this reasoning, a unitary diagram is consistent if and only if one or more of its γ components is consistent. The definition of consistency leads directly to a decision procedure which allows us to judge the satisfiability of a unitary diagram by inspecting its syntax.

3.1 Satisfiability for unitary diagrams

First, we introduce the class of diagrams called γ-diagrams. As we explain below, β-diagrams [28], in which each zone is shaded or contains a spider, or both, make it possible to identify inconsistency simply by observing the presence or absence of spiders. Our definition of inconsistency depends on β-diagrams, but also requires that all arrows are sourced on contours. For this reason we define γ-diagrams, which are β-diagrams in which every arrow is sourced on a contour. For convenience in definitions and results, we also define γ-diagrams to contain no non-trivial ties between spiders. We will show that for every α-diagram, d, we can construct a set of γ-diagrams, Γ, where d is satisfiable if and only if one of the diagrams in Γ is satisfiable.

Definition 3.1.1. Let d_1 be a generalized unitary diagram in which each zone is shaded, or contains a spider, or both. Then we say that d_1 is a **β-diagram**. Let d_2 be a generalized unitary β-diagram which satisfies the following:

1. each arrow $(l, s, t) \in A(d_2)$ is sourced on a contour, i.e. $s \in C(d)$, and
2. $\tau_{d_2} = \{(x, x) : x \in S(d_2)\}$.

Then we say that d_2 is a **γ-diagram**.

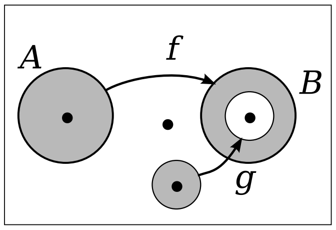


Figure 3.1: A γ-diagram.

The diagram in Figure 3.1 is a β-diagram since all zones are shaded or contain a contour, or both, and is also a γ-diagram since all arrows are sourced on contours.

3.1.1 Inconsistency

There are two possible causes of inconsistency in a generalized unitary diagram d. The first is its arrows and their potential to represent a situation which could not be reflected in the relation represented by the label of the arrow. The second is the case where spiders inhabiting distinct zones are joined by ties. Figure 3.2 presents three examples of inconsistency which may be present in a diagram,

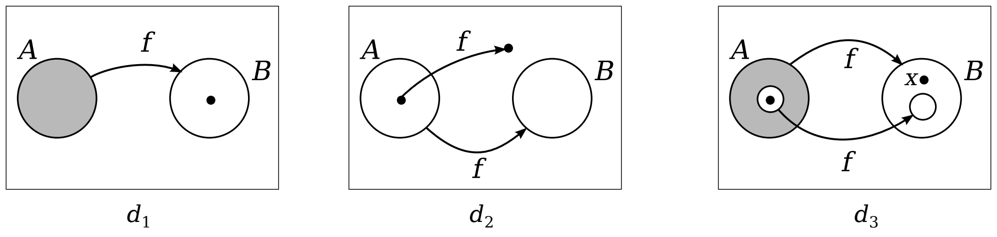


Figure 3.2: Three examples of inconsistency arising from arrows.

arising from its arrows. In d_1 inconsistency arises because there are too few elements in the set represented by the source of an arrow for the property of the relation it represents to hold. In particular, d_1 asserts

1. $\Psi(A) = \emptyset$,
2. $\Psi(B) \neq \emptyset$, and
3. $\Psi(A).\Phi(f) = \Psi(B)$.

These three statements cannot be true at the same time, so no interpretation satisfies d_1. In d_2 the arrow (f, A, B) also tells us that the image of f, when the domain has been restricted to A, is equal to B. The arrow sourced on the spider within A has a target spider outside of B, and so the two arrows are incompatible with each other. We have, informally, $A.f = B$ and that there is an element, a, in A such that $a.f$ is an element outside of B.

The third type of inconsistency, illustrated in d_3, is a more general form of that in d_1. It also arises when there are too few elements in the source set of an arrow to satisfy the constraint imposed by the arrows on the represented relation. The arrow (f, A, B) tells us that elements in A are related under f to all elements of B and to nothing else. However, there is no element of A that could possibly be related to the spider x. The subset, S, of A, represented by the derived contour inside A contains elements which are related to elements of B, but not to x. Since $S = A$, we have a contradictory situation. We informally give the name *source inconsistency* to the type of inconsistency depicted in d_1 and d_3; there are too few spiders at the source of an arrow. We call the inconsistency depicted in d_2 *target inconsistency*; there are arrows whose targets provide inconsistent information. We note that target inconsistency generalises the notion of *pairwise-*

incompatibility found in [43], and that source inconsistency arises because of the presence of arrows sourced on contours; this type of arrow did not occur in [43].

As a further example, Figure 3.3 shows a situation in which both forms of inconsistency arise because of the interaction of a pair of arrows. Target inconsistency arises because two arrows with the same label and source set of spiders have disjoint target sets of spiders. Source inconsistency arises because, for each of the arrows, there is a spider in the target set which must be related to some spider in the source set, but no spider is available.

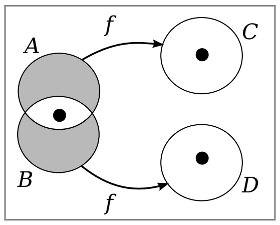


Figure 3.3: Source and target inconsistency arising together.

In essence, the properties of source and target inconsistency both arise due to the location or absence of spiders with respect to the arrows of the diagram. In fact, their definitions can be unified by considering how to identify inconsistency via the spiders in the targets and sources of arrows, relative to other arrows. In Figure 3.3, the arrow (f, A, C) has a spider, y, in its target. There is a spider, x, inhabiting a zone in the source of (f, A, C), but x cannot be related to y because it is in the source of another arrow, (f, B, D), with a target that is disjoint from that of (f, A, C). We can identify this lack of spiders in the source of (f, A, C) with respect to (f, B, D) by noting that if we subtract the spiders in the target of (f, B, D) from those in the target of (f, A, C) there are spiders remaining, but if we subtract the spiders in the source of (f, B, D) from those in (f, A, C), no spiders remain. Semantically speaking, there is no element in the set represented by A that can be related to an element in the set represented by C, although the arrow (f, A, C) tells us that this is so. To formulate general definitions, we focus on pairs of sets of arrows, rather than pairs of individual arrows. In this way we can identify the simplest form of source inconsistency shown in Figure 3.2, d_1, by considering the sets $X = \{(f, A, B)\}$ and $Y = \emptyset$. If we subtract the spiders in the targets of all arrows in Y from those in X, there are spiders remaining. If we carry out the same process for the sources of Y and X, there are no spiders, and d_1 is inconsistent.

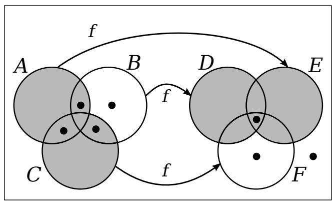


Figure 3.4: Interaction of multiple arrows.

By considering pairs of sets of arrows, we can account for more complex cases where multiple arrows interact. In Figure 3.4, take the sets $X = \{(f, C, F)\}$ and $Y = \{(f, A, E)), (f, B, D)\}$. There are spiders in F, the target of the only arrow in X, which are not in D or E, the targets of the arrows in Y. However, all spiders in C, the source of the only arrow in X, are also in the source of an arrow in Y, and so the diagram is inconsistent. Returning to the notion of pairwise-compatibility, in [43], Stapleton derives a relationship between a pair of arrows that identifies when inconsistency arises. In our system, a pairwise consideration of arrows is not sufficient to identify target inconsistency. For example, in Figure 3.4, no pair of the three arrows provide inconsistent information (and are not, therefore, pairwise-compatible), but the three arrows together do provide inconsistent information. We now make definitions to select the source and target spiders of a set of arrows.

Definition 3.1.2. Let d be a generalized unitary diagram and let $X \subseteq A(d)$. We define the **source spiders** in X, denoted $S_s(X, d)$, as follows.

$$S_s(X, d) = \bigcup_{(l,s,t)\in X} S(s, d).$$

Similarly, we define the **target spiders** of X, denoted $S_t(X, d)$, as follows.

$$S_t(X, d) = \bigcup_{(l,s,t)\in X} S(t, d).$$

When we compare the target and source spiders of sets of arrows, we are only interested in sets of arrows with the same label; inconsistency cannot arise by the interaction of arrows with different labels. Thus, we define the arrows with a given label in a diagram.

Definition 3.1.3. Let d be a generalized unitary diagram and let l be an arrow label in $AL(d)$. Define the **arrows of** l in d, denoted $A(l, d)$, as follows.

$$A(l, d) = \{(l', s, t) \in A(d) : l' = l\}.$$

So far we have illustrated how to use pairs of sets of arrows to identify source inconsistency, arising because there are too few elements in the source of an arrow (as in Figure 3.4). Sets of arrows are also sufficient to identify target inconsistency, arising because two arrows which share spiders in their source have disjoint targets. The diagram in Figure 3.5 illustrates target inconsistency. Set $X = \{(f, B, D)\}$ and $Y = \{(f, A, C)\}$. Then $S_t(X, d) - S_t(Y, d) \neq \emptyset$, but because all spiders in B are also in A, $S_s(X, d) - S_s(Y, d) = \emptyset$. Pairs of sets of arrows are thus sufficient to identify both forms of inconsistency. In fact, although it intuitively seems that there are two forms of inconsistency in generalized unitary diagrams, our method reveals that the two forms are manifestations of the same phenomenon.

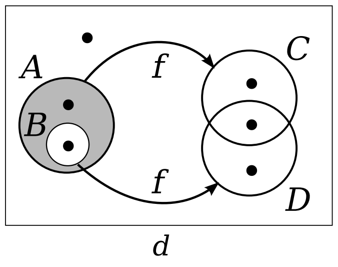


Figure 3.5: Target inconsistency.

In terms of choosing the sets of arrows to be compared, it is only necessary that they be disjoint. If, in Figure 3.5, we choose sets which are not disjoint, $X = \{(f, B, D)\}$ and $Y = \{(f, B, D), (f, A, C)\}$, inconsistency is not identified. The examples in this section have demonstrated that the conditions of a γ-diagram allow us to identify inconsistency by observing the spiders of a diagram relative to its arrows; Figure 3.6 demonstrates that β-diagrams are not sufficient. If we take $X = \{(f, A, B)\}$ and $Y = \{(f, x, y)\}$, then $S_t(X, d) - S_t(Y, d) \neq \emptyset$, but

$S_s(X, d) - S_s(Y, d) = \emptyset$. The diagram is thus judged to be inconsistent by our approach, although it is satisfiable. The use of γ-diagrams avoids this problem.

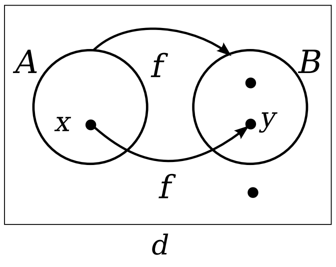


Figure 3.6: The need for γ-diagrams.

Definition 3.1.4. Let d ($\neq \perp$) be a generalized unitary γ-diagram. The diagram d is **consistent** if, for all arrow labels l in $AL(d)$ and for all pairs of sets of arrows X and Y in $A(l, d)$, where $X \cap Y = \emptyset$, the following holds:

$$S_t(X, d) - S_t(Y, d) \neq \emptyset \Rightarrow S_s(X, d) - S_s(Y, d) \neq \emptyset.$$

If this property does not hold we say d is **inconsistent**.

To show that the above definition is correct, we will show that every consistent diagram is satisfiable and every satisfiable diagram is consistent. Before we do, we establish a result relating to the source spiders of sets of arrows.

Lemma 3.1.1. Let d ($\neq \perp$) be a satisfiable γ-diagram and let X and Y be two disjoint subsets of $A(d)$. If

$$S_s(X, d) - S_s(Y, d) = \emptyset,$$

then

$$\bigcup_{(l,s,t)\in X} \Psi'(s) - \bigcup_{(l,s,t)\in Y} \Psi'(s) = \emptyset,$$

where (U, Ψ, Φ) is any model for d and Ψ' is a valid extension.

Proof. Assume

$$S_s(X, d) - S_s(Y, d) = \emptyset.$$

By the definition of S_s this tells us that

$$\big(\bigcup_{(l,s,t)\in X} S(s,d)\big) - \big(\bigcup_{(l,s,t)\in Y} S(s,d)\big) = \emptyset. \tag{3.1}$$

Let (l,s,t) be an arrow in X. By (3.1),

$$S(s,d) \subseteq \bigcup_{(l,s',t')\in Y} S(s',d). \tag{3.2}$$

Suppose $\Psi'(s) \neq \emptyset$ and let $e \in \Psi'(s)$. Then $e \in \Psi'(in, out)$ for some unique zone $(in, out) \in Z(d)$ where $s \in in$. Since d is a γ-diagram, (in, out) contains a spider, say x. Now,

$$S((in, out), d) \subseteq S(s,d)$$

and, by (3.2), we can deduce

$$x \in S((in, out), d) \subseteq S(s,d) \subseteq \bigcup_{(l,s',t')\in Y} S(s',d).$$

That is, $x \in \bigcup_{(l,s',t')\in Y} S(s',d)$. Choose some arrow, $(l,s',t') \in Y$ with $x \in S(s',d)$. Then it can be shown that $s' \in in$. Recall that $e \in \Psi'(s)$. Then, since (l,s,t) was an arbitrary element of X, for any $e \in \bigcup_{(l,s,t)\in X} \Psi'(s)$, it follows that

$$e \in \bigcup_{(l,s',t')\in Y} \Psi'(s').$$

Hence,

$$\bigcup_{(l,s,t)\in X} \Psi'(s) \subseteq \bigcup_{(l,s',t')\in Y} \Psi'(s')$$

and therefore

$$\bigcup_{(l,s,t)\in X} \Psi'(s) - \bigcup_{(l,s',t')\in Y} \Psi'(s') = \emptyset.$$

□

Now we will show that the first aspect of the relationship between consistency and satisfiability holds: that inconsistent diagrams are unsatisfiable.

Theorem 3.1.1: Let d ($\neq \perp$) be a generalized unitary γ-diagram. If d is inconsistent then d is unsatisfiable.

Proof. Assume d is inconsistent but satisfiable, and that (U, Ψ, Φ) is a model for d where (U, Ψ', Φ) is a valid extension. Since d is inconsistent, for some arrow label $l \in AL(d)$, there exist two sets of arrows labelled l, X and Y where the following is true:

1. $X \cap Y = \emptyset$,
2. $S_t(X, d) - S_t(Y, d) \neq \emptyset$,
3. $S_s(X, d) - S_s(Y, d) = \emptyset$.

Let y be a spider in $S_t(X, d) - S_t(Y, d)$ and choose an arrow (l, s', t') in X where $y \in S(t', d)$. In Figure 3.7 we use a meta-diagram to illustrate this condition, where the curves labelled with $\cup$ operators represent all sources and, respectively, targets of arrows in Y and details such as other contours in X are suppressed to focus on (l, s', t'). From the spiders habitat and the arrows condition we know that

$$\Psi'(y) \subseteq \Psi'(s').\Phi(l). \tag{3.3}$$

Then there exists some universal element e in U where

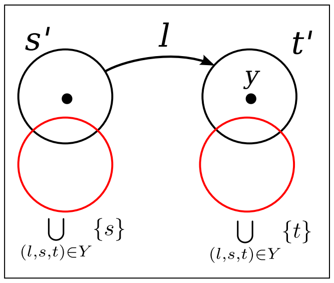

s'
l
t'
y
$\bigcup_{(l,s,t)\in Y} \{s\}$
$\bigcup_{(l,s,t)\in Y} \{t\}$

Figure 3.7: A spider in the target of one of the arrows of X.

$$e \in \Psi'(s') \tag{3.4}$$

and, by (3.3), $(e, \Psi'(y)) \in \Phi(l)$; see Figure 3.8. Therefore,

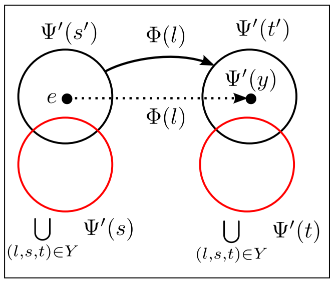

$\Psi'(s')$
$\Phi(l)$
$\Psi'(t')$
$\Psi'(y)$
e
$\Phi(l)$
$\bigcup_{(l,s,t)\in Y} \Psi'(s)$
$\bigcup_{(l,s,t)\in Y} \Psi'(t)$

Figure 3.8: The semantic implication of Figure 3.7.

$$e \in \bigcup_{(l,s',t')\in X} \Psi'(s'). \tag{3.5}$$

We know that $\Psi'(y) \not\subseteq \Psi'(s'').\Phi(l)$ for each $(l, s'', t'') \in Y$ since $y \notin S_t(Y, d)$ and the distinct spiders, spiders' habitat and arrow conditions hold (see for example Figure 3.7, where y lays outside $\bigcup_{(l,s,t)\in Y} \{t\}$). Therefore $e \notin \Psi'(s'')$ for each $(l, s'', t'') \in Y$, since nothing in the source of an arrow in Y is related to y. Hence,

$$e \notin \bigcup_{(l,s'',t'')\in Y} \Psi(s''). \tag{3.6}$$

By (3.5) and (3.6),

$$e \in \bigcup_{(l,s',t')\in X} \Psi'(s') - \bigcup_{(l,s'',t'')\in Y} \Psi(s'').$$

However, by lemma 3.1.1,

$$\bigcup_{(l,s',t')\in X} \Psi'(s') - \bigcup_{(l,s'',t'')\in Y} \Psi(s'') = \emptyset,$$

reaching a contradiction and showing that, if d is inconsistent then d is unsatisfiable. □

In the next section we show the reverse of the above result, i.e. that satisfiable diagrams are consistent.

3.1.2 Standard interpretations

The standard interpretation of a unitary α-diagram d is one in which the universe is given as the set of spiders of d and Ψ and Φ are 'natural' mappings to the universe and to relations on its elements, analogous to canonical models in classical logic [2]; in [43] this approach gives rise to a model for any satisfiable α-diagram. Due to the inclusion of arrows sourced on contours in our system we can only, in general, construct such a model for the class of γ-diagrams. Indeed, it can be shown that, in general, using $S(d)$ as the universal set of an interpretation, when attempting to construct a model for a unitary α-diagram, d, $S(d)$ may not have sufficiently large cardinality even though d may be satisfiable. We will illustrate the issues that arise via examples.

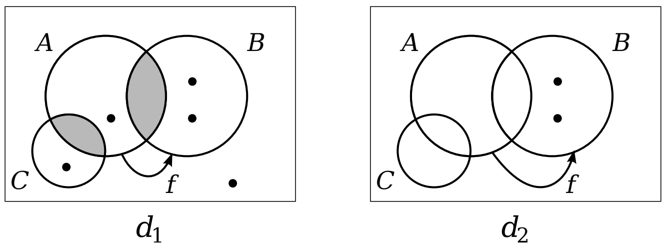


Figure 3.9: Constructing a model with too few source spiders.

In Figure 3.9, d_1, which is a γ-diagram, has sufficient spiders in the source of the only arrow and we can construct a model (U, Ψ, Φ) for d_1 as follows.

1. $U = S(d_1)$,
2. $\Psi(A) = S(A, d_1)$,
3. $\Psi(B) = S(B, d_1)$,
4. $\Psi(C) = S(C, d_1)$,
5. $\Psi(c) = \emptyset$ for all other contour labels $c \in \mathcal{GC}$.
6. $\Phi(f) = \{(x, t_1), (x, t_2)\}$, where $x \in S(A, d_1)$, $t_1, t_2 \in S(B, d_1)$ and $t_1 \neq t_2$,
7. $\Phi(l) = \emptyset$ for all other arrow labels $l \in \mathcal{AL}$.

It is clear that this is a minimal model for d_1 since no interpretation based on a smaller universe would suffice (we require $|U|$ to be at least $|S(d)|$). The approach runs into complications when we consider diagrams such as d_2, Figure 3.9. In this diagram, there are too few spiders in the source of the arrow (f, A, B) to satisfy the relation represented by f when its domain is restricted to the set represented by A. Since A contains an unshaded zone, d_2 is satisfiable, but in an interpretation for d_2 which uses the set of spiders as the universe, the spiders' habitat and arrows conditions cannot be true at the same time. In the metatheory, it is extremely helpful to be able to produce a minimal model for any diagram. Creating a procedure to do so is not straightforward when the set of spiders is too small to serve as the universe. We therefore make use of γ-diagrams. This constraint ensures that no contour is 'lacking spiders' for any arrow, i.e. no arrow has too few source spiders, ruling out the situation in Figure 3.9, diagram d_2. γ-diagrams allow us to define a standard interpretation in which the universe is the set of spiders of the diagram and Ψ maps contours, regions and zones to subsets of the set of spiders. This is because if we have an arrow, say (l, s, t), where t contains spiders, it is necessary that s contains spiders; informally, the arrow tells us $s.l = t \neq \emptyset$. More complicated situations can arise that force the existence of elements in zones that do not contain spiders. Thus, in defining γ-diagrams we extend the definition of β-diagrams, in which each zone is either shaded or contains a spider.

When constructing the mapping, Φ, for arrow labels, it may be tempting to map every spider in the source set of an arrow to its set of target spiders, but Figure 3.10 illustrates the need to restrict the elements in the domain and image of the mapping. In d_1 the arrow (f, A, B) requires that, when constructing a model, every element of $\Psi(B)$ is included in $\Psi(A).\Phi(f)$. Naively mapping every element of A to every element of B would produce a diagram which included the relationships as depicted by the arrows depicted in d_2, and such a mapping cannot arise in an interpretation that satisfies d_1.

Definition 3.1.5. Let d $(\neq \perp)$ be a generalized γ-diagram. Let $m = (U, \Psi, \Phi)$ be an interpretation defined as follows.

1. $U = S(d)$,

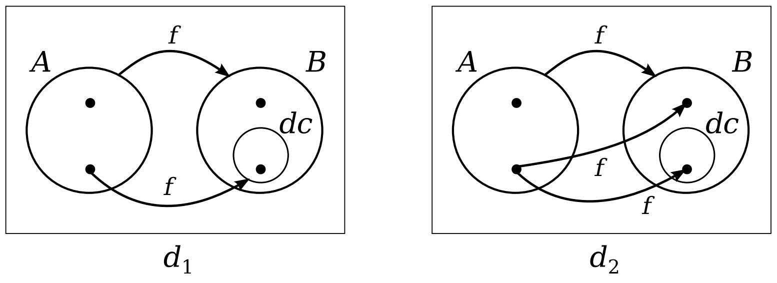


Figure 3.10: Restricting the mapping for arrow labels.

2. For all contours c in $\mathcal{GC}$,

$$\Psi(c) = \begin{cases} S(c,d) & \text{if } c \in C(d)\,, \\ \emptyset & \text{otherwise.} \end{cases}$$

3. For all labels $l \in \mathcal{AL}$,

$$\Phi(l) = (S(d) \times S(d)) - \{(x,y) : \exists (l,s,t) \in A(d) \\ (x \in S(s,d) \wedge y \notin S(t,d))\}.$$

Define m as the **standard interpretation** for d.

The definition of Φ gives a 'greedy' mapping, in that it will relate all universal elements whenever doing so does not give rise to a contradiction.

Definition 3.1.6. Let d ($\neq \bot$) be a generalized γ-diagram and let $m = (U, \Psi, \Phi)$ be the standard interpretation for d. To form the **standard extension** for the spiders and derived contours of d we extend Ψ to Ψ' where

$$\Psi' : \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S} \to \mathbb{P}U,$$

where for each spider x, $\Psi'(x) = \{x\}$ if $x \in S(d)$, and $\Psi'(x) = \emptyset$ otherwise. Furthermore, for each derived contour dc in $DC(d)$,

$$\Psi'(dc) = S(dc, d).$$

We now establish that the mapping of zones to sets of spiders behaves as we expect: $\Psi'(z) = S(z, d)$.

Lemma 3.1.2. Let d ($\neq \bot$) be a generalized unitary γ-diagram, let (U, Ψ, Φ) be the standard interpretation for d and let Ψ' be the standard extension for d. Then, for all zones $z \in Z(d)$,

$$\Psi'(z) = S(z, d).$$

Proof. Let (in, out) be a zone in $Z(d)$. In the definition of Ψ' we have

$$\Psi'(in, out) = \bigcap_{c \in in} \Psi'(c) \cap \bigcap_{c \in out} (U - \Psi'(c)). \tag{3.7}$$

Since m is the standard extension, for each of the contours $c \in in$ we have, by definition,

$$\Psi'(c) = S(c, d). \tag{3.8}$$

We can rewrite (3.7) using (3.8) as follows:

$$\Psi'(in, out) = \bigcap_{c \in in} S(c, d) \cap \bigcap_{c \in out} (U - S(c, d)). \tag{3.9}$$

Since $U = S(d)$, this becomes

$$\Psi'(in, out) = \bigcap_{c \in in} S(c, d) \cap \bigcap_{c \in out} (S(d) - S(c, d)).$$

Let x be a spider in $S((in, out), d)$. Then, for each $c \in in$, $x \in S(c, d)$ since, by the definition of $S(c, d)$, we have

$$S(c, d) = \{x' \in S(d) : c \in in \text{ where } \eta_d(d) = (in, out)\}.$$

So,

$$x \in \bigcap_{c \in in} S(c, d).$$

Similarly, for any $c \in out$, $x \notin S(c, d)$ and, therefore, $x \in S(d) - S(c, d)$. Thus,

$$x \in \bigcap_{c \in out} (S(d) - S(c, d)).$$

By 3.9 we have

$$x \in \bigcap_{c \in in} S(c,d) \cap \bigcap_{c \in out} (S(d) - S(c,d)) = \Psi'(in, out).$$

Therefore each spider in $S((in, out), d)$ is also in $\Psi'(in, out)$ and we have

$$S((in, out), d) \subseteq \Psi'(in, out).$$

The proof that $\Psi'(in, out) \subseteq S((in, out), d)$ is similar. Hence, for all zones, z, in $Z(d)$, $\Psi'(z) = S(z, d)$. $\square$

We are now ready to show that the standard interpretation for a γ-diagram d is indeed a model for d.

Theorem 3.1.2: Let d ($\neq \perp$) be a generalized unitary γ-diagram that is consistent and let $m = (U, \Psi, \Phi)$ be the standard interpretation for d. Then m is a model for d and the standard extension for d, $m' = (U, \Psi', \Phi)$, is valid.

Proof. We show that m satisfies d by considering the semantic formula given m'. The plane tiling condition states that $\Psi'(Z) = U$. With regards to m', we know that $U = S(d)$ and, by lemma 3.1.2, for all zones z in $Z(d)$,

$$\Psi'(z) = S(z, d).$$

Therefore, since each spider is in some zone of d,

$$\Psi'(Z(d)) = \bigcup_{z \in Z(d)} S(z, d) = S(d) = U.$$

The shaded zones condition holds by a similar argument. The spiders' habitat and distinctness conditions are trivially true, as can be seen from the definition of U and Ψ'. We next show that the arrows condition in m holds for d. Let (l, s, t) be an arrow in $A(d)$. We wish to show that $\Psi'(s).\Phi(l) = \Psi'(t)$, which by the definition of m' is equivalent to $S(s, d).\Phi(l) = S(t, d)$. We do this by showing that

(1) $S(s, d).\Phi(l) \subseteq S(t, d)$,

(2) $S(t,d) \subseteq S(s).\Phi(l,d)$.

We begin by showing (1) is true. Choose a spider, y, in $S(s,d).\Phi(l)$. Then there exists a spider, x, in $S(s,d)$ and $(x,y) \in \Phi(l)$. Suppose $y \notin S(t,d)$. But the arrow we are considering, (l,s,t), has $x \in S(s,d)$ and $y \notin S(t,d)$. By the definition of Φ, the standard interpretation would not relate x and y under l, that is $(x,y) \notin \Phi(l)$, reaching a contradiction and showing that $y \in S(t,d)$ and hence $S(s,d).\Phi(l) \subseteq S(t,d)$.

We now show (2) is true. Let $y \in S(t,d)$. We will show that there exists an x such that $(x,y) \in \Phi(l)$ and $x \in S(s,d)$. Let $X_1 = \{(l,s,t)\}$ and $Y_1 = \emptyset$. We know that there are spiders in $S_t(X_1,d) - S_t(Y_1,d)$ (i.e $y \in S_t(X_1,d) - S_t(Y_1,d)$) so, since d is consistent, we know that $S_s(X_1,d) - S_s(Y_1,d)$ is not empty, and hence $S(s,d) \neq \emptyset$. Suppose no x in $S(s,d)$ is related to y. That is, for all $x' \in S(s,d)$, $(x',y) \notin \Phi(l)$. Then, by the definition of $\Phi(l)$, for each $x' \in S(s,d)$ there is an arrow whose source includes x' but whose target does not include y:

$$\forall x' \in S(s,d)\, \exists\, (l,\hat{s},\hat{t})\, (x' \in S(\hat{s},d) \wedge y \notin S(\hat{t},d)). \tag{3.10}$$

Define $X_2 = \{(l,s,t)\}$ and Y_2 to be the set of arrows $(l,\hat{s},\hat{t})$ where there is some $x' \in S(s,d)$, for which $x' \in S(\hat{s},d)$ and $y \notin S(\hat{t},d)$ (that is, arrows as in (3.10)). Then

$$y \in S_t(X_2,d) - S_t(Y_2,d),$$

but

$$S_s(X_2,d) - S_s(Y_2,d) = \emptyset.$$

Thus, d is inconsistent, but this is a contradiction since we know d is consistent by assertion. Therefore, there exists $x \in S(s,d)$ where $(x,y) \in \Phi(l)$. Hence, $S(t,d) \subseteq S(s,d).\Phi(l)$. It follows that $\Psi(s).\Phi(l) = \Psi(t)$, the arrows condition holds for d and the semantic formula, *form*(d), is true. Hence, m is a model for d and the standard extension is valid. □

3.1.3 A decision procedure for the satisfiability of unitary γ-diagrams

We are now able to show that a unitary γ-diagram is satisfiable if and only if it is consistent, thus providing a decision procedure for the satisfiability of unitary γ-diagrams.

Theorem 3.1.3: Let d ($\neq \perp$) be a generalized unitary γ-diagram. The diagram d is satisfiable if and only if d is consistent. Furthermore, the unitary γ fragment is decidable.

Proof. Since d is a γ-diagram, it is also an α-diagram. Therefore, by theorem 3.1.1, if d is inconsistent then d is unsatisfiable.

By lemma 3.1.2, if d is not inconsistent then d is satisfiable via the standard interpretation.

Since d contains finitely many spiders, contours and arrows, we can determine the satisfiability of d by checking whether target inconsistency arises for any pair of arrows, and whether source inconsistency arises for any arrow sourced on a contour. Hence, the unitary γ fragment is decidable. $\square$

We are now required to show that we can use the class of γ-diagrams as a reduction class for the unitary existential fragment. That is, we must show that any diagram from the unitary existential fragment is equivalent to some set of γ-diagrams, thus enabling us to use the above decision procedure with any unitary diagram. To do so, we will define a series of inference rules. To facilitate elegant definitions of inference rules for unitary and, later, non-unitary generalized diagrams, we will base the definitions on *transformations*.

3.2 Transformations

Transformations are syntactic operations which represent the addition or removal of a piece of syntax. For example, we can remove the curve B from d_1 in Figure 3.11, transforming it into d_2; this remove curve transformation will be formalised below.

The transformations defined will be applicable under specified syntactic conditions, which are not related to sound reasoning, but are intended to merely

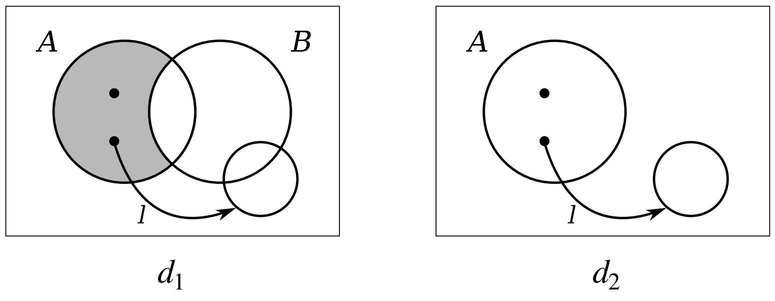


Figure 3.11: Purely syntactic transformations.

constrain the transformation to ensure the result of its application is a diagram. The benefit of making transformations which are purely syntactic and unrelated to reasoning is that this facilitates their use in a wide number of (reasoning) contexts. We discuss the benefits of the transformations approach at more length in section 5.1.

Figure 3.12 shows an invalid application of a transformation which removes a spider, which erases a spider that is the source of an arrow. The invalid transformation results in the collection of lines and curves labelled d_2, which is not a generalized unitary diagram since it includes an arrow with no source.

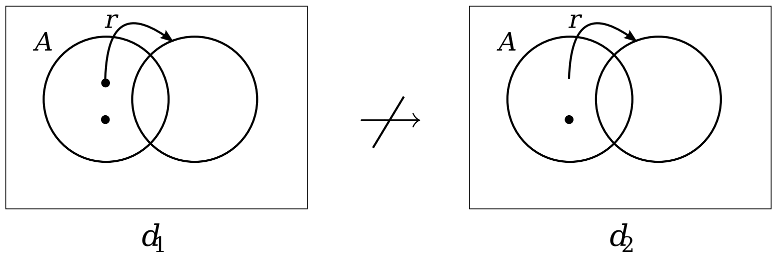


Figure 3.12: An invalid transformation.

Any conditions on the removal or addition of syntax which relate to reasoning, such as that the habitat of a spider to be removed must be unshaded, are introduced by associated inference rules. In general, inference rules combine the use of one or more transformations with an additional set of pre and post conditions. In order to maximise the usefulness of the transformations at the reasoning stage, we aim for transformations defined in a general way and with minimal preconditions. If a transformation T can be applied to a diagram d_1 to give d_2, we write

$d_1 \xrightarrow{T} d_2$.

Transformations may rely on each other, in that a transformation T may be applicable only after other transformations have been applied. For instance, an arrow can always be removed from a diagram to give a second, well-formed diagram, but a spider may only be removed if it is not the source or target of an arrow. Removing a spider may therefore require the removal of arrows sourced on, or targeting it. We introduce the transformations with weakest preconditions first and simplify some transformations by adding preconditions which can be satisfied by the application of other transformations.

3.2.1 Transformations that remove syntax

Transformation 1: Remove arrow. We can transform a diagram by removing an arrow, including arrows which touch derived contours. In Figure 3.13, the arrow, a, labelled r is removed from d_1 to give d_2.

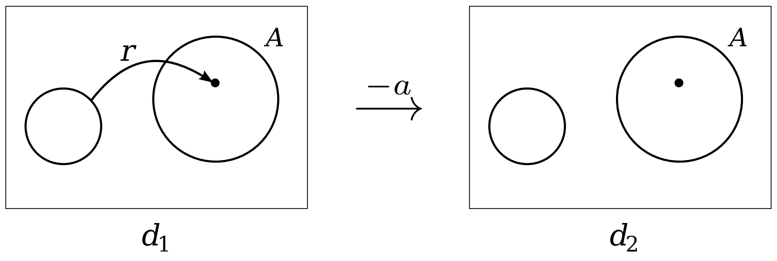


Figure 3.13: Transforming a diagram by removing an arrow.

Formal description. Let d_1 be a generalized unitary diagram such that there exists an arrow $a \in A(d_1)$. The diagram d_2 can be obtained from d_1 by removing a, denoted $d_1 \xrightarrow{-a} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1), S(d_1), \eta_{d_1}, \tau_{d_1}, A(d_1) - \{a\}).$$

Transformation 2: Remove shading. We can transform a diagram by removing the shading from a zone, without any preconditions. In Figure 3.14, the shading is removed from the zone $(\{B\}, \{A\})$ in d_1 to give d_2.

Formal description. Let d_1 be a generalized unitary diagram and let z be a zone such that $z \in Z^*(d_1)$. The diagram d_2 can be obtained from d_1 by using

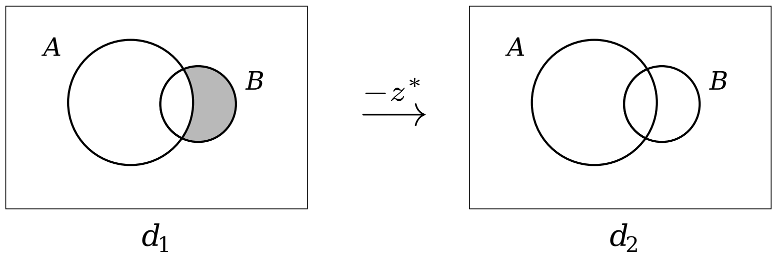


Figure 3.14: Transforming a diagram by removing shading from a zone.

the **remove shading** transformation to remove the shading from z, denoted $d_1 \xrightarrow{-z^*} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1) - \{z^*\}, S(d_1), \eta_{d_1}, \tau_{d_1}, A(d_1)).$$

Transformation 3: Remove spider. We can transform diagrams by removing a spider as long as the spider is not the source or target of an arrow. In Figure 3.15 the spider labelled x is removed from d_1 to give d_2. When removing a spider, x, from a diagram, d, we need to ensure that the equality relation τ_d is updated to remove pairs which include x.

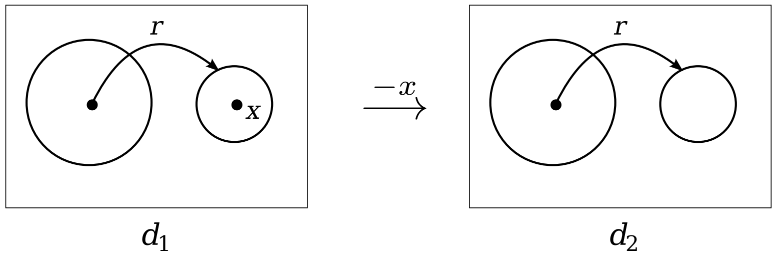


Figure 3.15: Transforming a diagram by removing a spider.

Formal description. Let d_1 be a generalized unitary diagram such that there exists a spider $x \in S(d_1)$ which is not the source or target of any arrow. The diagram d_2 can be obtained from d_1 by using the **remove spider** transformation to remove x, denoted $d_1 \xrightarrow{-x} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1), S(d_1) - \{x\}, \eta_{d_1}|_{S(d_1)-\{x\}}, \tau_{d_2}, A(d_1)),$$

and

$$\tau_{d_2} = \{(x', y') \in \tau_{d_1} : x' \neq x \wedge y' \neq x\}.$$

When removing ties between spiders in unitary diagrams, we make changes to the equality relation, τ, of the diagram. In order to do this, we define notation used to denote the equivalence class of each spider under τ.

Definition 3.2.1. Let d be a generalized unitary diagram and let $x \in S(d)$. We define the equivalence class of x under τ_d, denoted $[x]_d$, as follows:

$$[x]_d = \{y : (x, y) \in \tau_d\}.$$

Transformation 4: Remove ties. We can transform diagrams by removing a tie between two spiders. In Figure 3.16, the tie between the spiders labelled x and y in diagram d_1 is removed to obtain d_2. If x and y are tied, as in d_1, x and y are members of the same equivalence class under τ. Removing the tie has the effect that x is no longer related to any member of $[y]_{d_1}$ except itself. For this reason, the transformation is called remove *ties*, rather than remove *tie*.

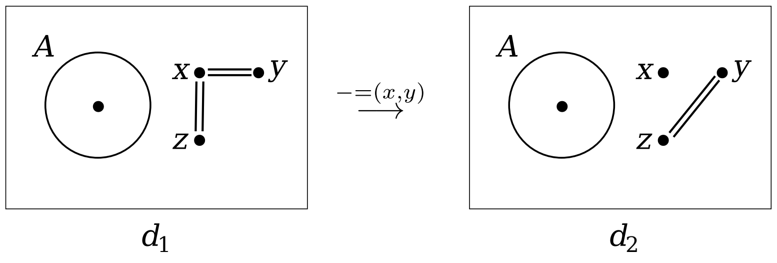


Figure 3.16: Transforming a diagram by removing a tie.

Formal description. Let d_1 be a generalized unitary diagram such that there exist two distinct spiders $x, y \in S(d_1)$ where $\tau_d(x, y)$. The diagram d_2 can be obtained from d_1 by the **remove ties** transformation, denoted $d_1 \xrightarrow{-=(x,y)} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1), S(d_1), \eta_{d_1}, \tau_{d_2}, A(d_1)),$$

and

$$\tau_{d_2} = \tau_{d_1} - \{(x, y'), (y', x) : y' \in [y]_{d_1} - \{x\}\}.$$

Removing the tie between two spiders, x and y, in a unitary diagram d can be thought of in terms of the equivalence classes under τ in d. The operation has the effect of splitting the class $[x]_d = [y]_d$ into two, $[y]_d - \{x\}$ and $\{x\}$. We are required to prove that the resulting relation is an equivalence relation.

Lemma 3.2.1. Let d_1 be a generalized unitary γ-diagram which has distinct spiders x and y, joined by a tie. Let d_2 be the diagram obtained by using the remove ties transformation to remove the tie between x and y. Then τ_{d_2} is an equivalence relation.

Proof. We know that τ_{d_1} is an equivalence relation since d_1 is a generalized unitary diagram. The relation τ_{d_2} is formed as follows:

$$\tau_{d_2} = \tau_{d_1} - \{(x, y'), (y', x) : y' \in [y]_{d_1} - \{x\}\}.$$

We know that (x, x) is in τ_{d_1}. Since the pairs removed from τ_{d_1} all have an element in $[y]_{d_1} - \{x\}$, (x, x) is in τ_{d_2}. For all $y' \in [y]_{d_1}$, the pair (y', y') is not removed, and so τ_{d_2} is reflexive. We will next show that τ_{d_2} is symmetric. Let $x', y' \in S(d_2) - \{x\}$. If $\tau_{d_2}(x', y')$ then we know that $\tau_{d_1}(x', y')$ since $\tau_{d_2} \subseteq \tau_{d_1}$. Then $\tau_{d_1}(y', x')$ since τ_{d_1} is symmetric. Since $x' \neq x$ and $y' \neq x$ it follows that

$$(y', x') \notin \{(x, y'), (y', x) : y' \in [y]_{d_1} - \{x\}\}. \tag{3.11}$$

Thus, we have $\tau_{d_2}(y', x')$ as required. Since τ_{d_2} is reflexive, $(x, x) \in \tau_{d_2}$ and symmetry is trivially true in this case. Thus τ_{d_2} is symmetric. To show transitivity, assume we have $x', y', z' \in S(d_1) - [x]_{d_1}$ where $(x', y') \in \tau_{d_1}$ and $(y', z') \in \tau_{d_1}$. Then we know that $(x', z') \in \tau_{d_1}$. We need to show that $(x', z') \in \tau_{d_2}$. As the pairs removed from τ_{d_1} all have an element in $[y]_{d_1} - \{x\}$ and $[x]_{d_1} = [y]_{d_1}$, no pair with an element from x', y' or z' is removed from τ_{d_1} to form τ_{d_2}. It follows that if the three elements were related in the way described in τ_{d_1} then the following is true:

1. $(x', y') \in \tau_{d_2}$,
2. $(y', z') \in \tau_{d_2}$, and
3. $(x', z') \in \tau_{d_2}$.

Suppose we have x'',y'' and z'' in $[y]_{d_1} - \{x\}$. By (3.11), only pairs containing x as an element are removed from τ_{d_1}, so (x'', y''), (y'', z'') and (x'', z'') are elements of τ_{d_2}. Thus τ_{d_2} is transitive, and is an equivalence relation as required. □

Transformation 5: Remove zone. We can transform a diagram by removing any zone which is not the habitat of any spider and so long as the resulting diagram contains the zone outside all others. In Figure 3.17 the zone $(\{A, B\}, \emptyset)$ is removed from d_1 to give d_2.

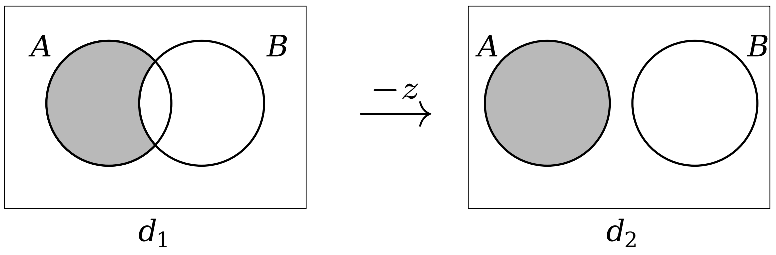


Figure 3.17: Transforming a diagram by removing a zone.

Formal description. Let d_1 be a generalized unitary diagram such that there exists a zone z in $Z(d_1)$ which is not the habitat of any spider and is not the zone outside all others, i.e. $S(z, d) = \emptyset$ and $z \neq (\emptyset, C(d_1))$. Then the diagram d_2 can be obtained from d_1 by using the **remove zone** transformation to remove z, denoted $d_1 \xrightarrow{-z} d_2$, where d_2 satisfies the following:

$$d_2 = (Z(d_1) - \{z\}, Z^*(d_1) - \{z\}, S(d_1), \eta_{d_1}, \tau_{d_1}, A(d_1)).$$

Before defining the transformation which removes a contour from a diagram, we define three sets of zones formed from the zones of $Z(d)$: those which were split by, inside and outside c.

Definition 3.2.2. Let d be a generalized unitary diagram and let $c \in C(d)$. We define three sets of zones formed by removing c from the zones of $Z(d)$ as follows.

1. $Z_s(c, d)$ is formed of the zones which were split by c:

$$\{(in, out) : (in \cup \{c\}, out) \in Z(d) \wedge (in, out \cup \{c\}) \in Z(d)\},$$

2. $Z_i(c,d)$ is formed of the zones which were inside c:

$$\{(in, out) : (in \cup \{c\}, out) \in Z(d) \wedge (in, out \cup \{c\}) \in MZ(d)\},$$

3. $Z_o(c,d)$ is formed of the zones which were outside c:

$$\{(in, out) : (in \cup \{c\} \in MZ(d) \wedge (in, out \cup \{c\}) \in Z(d)\}.$$

We define $Z^*_s(c,d)$, $Z^*_i(c,d)$ and $Z^*_o(c,d)$ similarly.

1. $Z^*_s(c,d)$ is formed of the zones which were inside fully shaded regions in d_1 which were split by c:

   $\{(in, out) : (in \cup \{c\}, out) \in Z^*(d) \wedge (in, out \cup \{c\}) \in Z^*(d)\},$

2. $Z^*_i(c,d)$ is formed of the zones which were shaded in d and inside c:

   $\{(in, out) : (in \cup \{c\}, out) \in Z^*(d) \cap Z_i(c,d)\}$

3. $Z^*_o(c,d)$ is formed of the zones which were shaded in d and outside c:

   $\{(in, out) : (in, out \cup \{c\}) \in Z^*(d_1) \cap Z_o(c,d)\}.$

Transformation 6: Remove contour. We can transform diagrams by removing contours so long as the contour is not touched by any arrow. In Figure 3.18 the contour labelled B is removed from d_1 to give d_2.

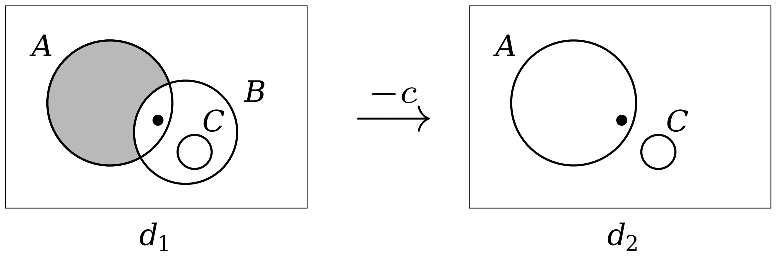


Figure 3.18: Transforming a diagram by removing a contour.

This transformation handles partial shading in the same way as the corresponding reasoning rules from notations such as Euler diagrams [44]. In diagram d_1 in Figure 3.18, the region inside the contour labelled A is partially shaded.

In the Euler diagram system, removing the contour B requires us to make the new region inside A entirely unshaded in order for the rule to be sound. The transformation is defined to work in the same way, since we intend to use it to make definitions of sound reasoning rules.

Formal description. Let d_1 be a generalized unitary diagram and let c be a contour such that $c \in C(d_1)$ and $A(c, d_1) = \emptyset$. The diagram d_2 can be obtained from d_1 by using the **remove contour** transformation to remove c, denoted $d_1 \xrightarrow{-c} d_2$, where d_2 satisfies

1. $Z(d_2) = \{(in - \{c\}, out - \{c\}) : (in, out) \in Z(d_1)\}$,
2. $Z^*(d_2)$ is the union of the sets of zones $Z^*_s(c, d_1)$, $Z^*_i(c, d_1)$ and $Z^*_o(c, d_1)$,
3. $S(d_2) = S(d_2)$,
4. $\forall s \in S(d_2)\ \eta_{d_2}(s) = (in - \{c\}, out - \{c\})$, where $(in, out) = \eta_{d_1}(s)$,
5. $\tau_{d_2} = \tau_{d_1}$,
6. $A(d_2) = A(d_1)$.

3.2.2 Transformations that add items of syntax

Transformation 7: Add arrow. Arrows can be added to a diagram in four ways, corresponding to the combinations of spiders and contours as source and target. We define a single add arrow transformation which can be used in any of the four ways. Figure 3.19 shows the addition of an arrow labelled r to the diagram d_1 to give d_2. The arrow is sourced on the contour labelled A and targets the spider whose habitat is $(\emptyset, \{A\})$.

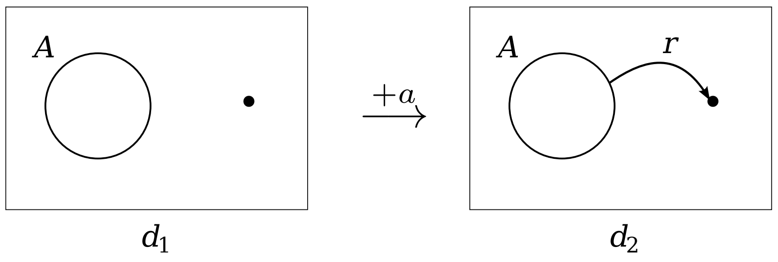


Figure 3.19: Transforming a diagram by adding an arrow.

Formal description. Let d_1 be a generalized unitary diagram and let (l, s, t) be an arrow such that $s, t \in S(d) \cup C(d)$. The diagram d_2 can be obtained by applying the **add arrow** transformation to add the arrow $a = (l, s, t)$ to d_1, denoted $d_1 \xrightarrow{+a} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1), S(d_1), \eta_{d_1}, \tau_{d_1}, A(d_1) \cup \{(l, s, t)\}).$$

Transformation 8: Add spider. We can transform a diagram by placing a new spider in a zone. Figure 3.20 shows the addition of a spider x, placed in the zone $(\emptyset, \{A, B\})$ to transform d_1 to d_2.

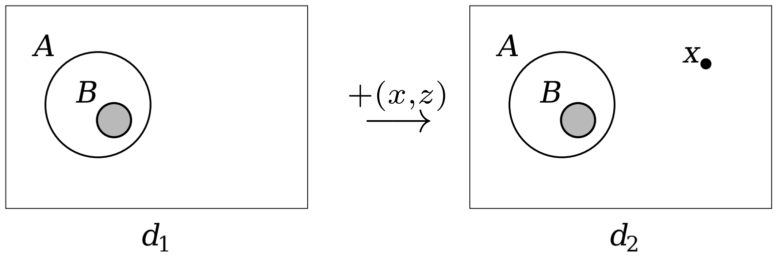


Figure 3.20: Transforming a diagram by adding an spider to a zone.

Formal description. Let d_1 be a generalized unitary diagram such that there exists a zone $z \in Z(d_1)$ and a spider $x \notin S(d_1)$. The diagram d_2 can be obtained by applying the **add spider** transformation to d_1, denoted $d_1 \xrightarrow{+(x,z)} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1), S(d_1) \cup \{x\}, \eta_{d_1} \cup \{(x, z)\}, \tau_{d_1} \cup \{(x, x)\}, A(d_1)).$$

Transformation 9: Add ties. We can transform a diagram by adding a tie between two spiders. In Figure 3.21, diagram d_2 shows the result of transforming d_1 by the addition of a tie between the spiders labelled x and y. The transformation is defined to preserve the properties of the equality relation, so that in d_2 not only are x and y related under τ_{d_2} but all members of the respective equivalence classes are related in such a way that transitivity and symmetry are preserved; reflexivity is maintained trivially. The add ties transformation has the effect of combining two equivalence classes of τ_{d_1}.

Formal description. Let d_1 be a generalized unitary diagram and let $x, y \in S(d)$

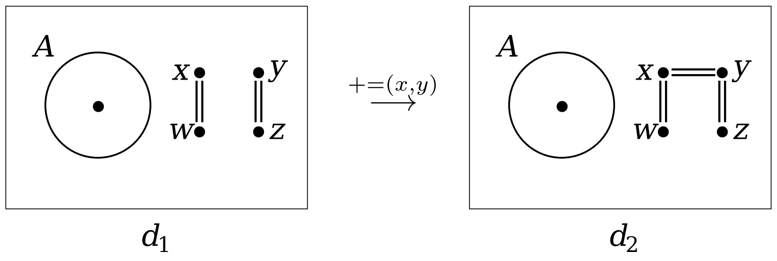


Figure 3.21: Transforming a diagram by adding a tie between two spiders.

such that $(x, y) \notin \tau_{d_1}$. The diagram d_2 can be obtained by applying the **add ties** transformation to d_1, denoted $d_1 \xrightarrow{+=(x,y)} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1), S(d_1), \eta_{d_1}, \tau_{d_2}, A(d_1)),$$

and τ_{d_2} is the symmetric, transitive closure of $\tau_{d_1} \cup \{(x, y)\}$.

Lemma 3.2.2. Let d_1 be a generalized unitary γ-diagram which contains spiders, x and y, such that $(x, y) \notin \tau_{d_1}$. Let d_2 be the diagram obtained by using the add ties transformation to add a tie between x and y. Then τ_{d_2} is an equivalence relation.

Proof. By the definition of the transformation, τ_{d_2} is the symmetric, transitive closure of $\tau_{d_1} \cup \{(x, y)\}$. Thus, we have

$$\tau_{d_2} = \tau_{d_1} \cup \bigcup_{x' \in [x]_{d_1}} \{(x', y'), (y', x') : y' \in [y]_{d_1}\}. \tag{3.12}$$

For each $x' \in S(d_2)$, we know that $x' \in S(d_1)$ and $(x', x') \in \tau_{d_1}$. Therefore, $(x', x') \in \tau_{d_2}$ and τ_{d_2} is reflexive. We know that τ_{d_1} is symmetric. By (3.12) we can clearly see that the set that forms $\tau_{d_2} - \tau_{d_1}$ forms a symmetric relation, and so τ_{d_2} is symmetric. For transitivity, let (x', y') and (y', z') be elements of τ_{d_2}. We need to show that $(x', z') \in \tau_{d_2}$. If (x', y') and (y', z') are elements of τ_{d_1} then we are finished, since τ_{d_1} is transitive. Assume $(x', y') \notin \tau_{d_1}$ and $(y', z') \in \tau_{d_1}$. Then either $x' \in [x]_{d_1}$ and $y' \in [y]_{d_1}$ or vice versa. Assume that $x' \in [x]_{d_1}$ and $y' \in [y]_{d_1}$. We know that $z' \in [y]_{d_1}$ since $(y', z') \in \tau_{d_1}$ and, by (3.12), $(x', z') \in \tau_{d_2}$. The other cases are similar. Thus, τ_{d_2} is transitive and is an equivalence relation as required. □

Transformation 10: Add shading. We can transform a diagram by adding shading to a zone. In Figure 3.22, shading is added to the zone $(\{A,C\},\{B\})$ in d_1 to give d_2.

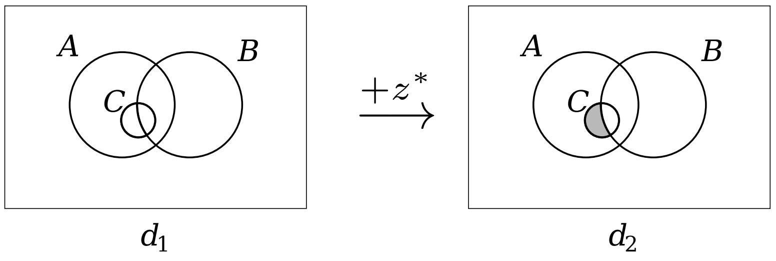


Figure 3.22: Transforming a diagram by adding shading to a zone.

Formal description. Let d_1 be a generalized unitary diagram and $z \in Z(d_1) - Z^*(d_1)$. Then the diagram d_2 can be obtained by applying the **add shading** transformation to d_1, denoted $d_1 \xrightarrow{+z^*} d_2$, where

$$d_2 = (Z(d_1), Z^*(d_1) \cup \{z\}, S(d_1), \eta_{d_1}, \tau_{d_1}, A(d_1)).$$

Transformation 11: Add zone. We can transform a diagram by adding a missing zone. Figure 3.23 shows the addition of the missing zone $(\{A,B\},\{C\})$ to d_1 to give d_2.

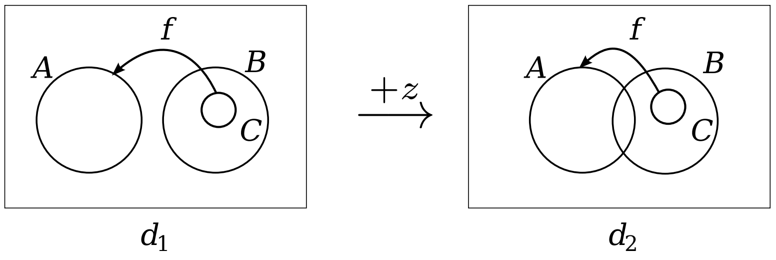


Figure 3.23: Transforming a diagram by adding a zone.

Formal description. Let d_1 be a generalized unitary diagram and z be a zone such that $z \in VZ(d_1) - Z(d_1)$. Then the diagram d_2 can be obtained by applying the **add zone** transformation to d_1, denoted $d_1 \xrightarrow{+z} d_2$, where

$$d_2 = (Z(d_1) \cup \{z\}, Z^*(d_1), S(d_1), \eta_{d_1}, \tau_{d_1}, A(d_1)).$$

Transformation 12: Add contour. There are a number of ways of adding a contour to a diagram in which other contours are present; the new contour can be added in such a way that it is entirely outside of all existing contours, or is entirely contained within one other contour, and so on. Previous systems have added contours in such a way that they split every existing zone. This is arguably an intuitive method and other arrangements of zones can be reached afterwards by the use of other reasoning rules. We can achieve greater flexibility, however, by parametrising the transformation of adding a contour to a diagram d_1 with two subsets of zones which we call Z_{in} and Z_{out}, where $Z_{in} \cup Z_{out} = Z(d_1)$. Those zones which will fall inside the new contour are in Z_{in}, those outside in Z_{out} and those that will be split are in $Z_{in} \cap Z_{out}$. Similarly, we require the user of the transformation to specify two sets of spiders, S_{in} and S_{out}, which are those spiders inside and outside of the new contour respectively, and where S_{in} and S_{out} partition the set of spiders of the diagram. An appropriate choice of Z_{in} and Z_{out} allows the user to add contours in any of the possible ways. The case of adding a contour to a diagram d which splits every zone in d_1, for instance, is that of choosing $Z_{in} = Z_{out} = Z(d)$. Figure 3.24 shows an example of adding a contour with $Z_{in} = \{(\emptyset, \{A, B\})\}$ and $Z_{out} = Z(d_1) - Z_{in}$, and with $S_{in} = \{x\}$ and $S_{out} = S(d_1) - \{x\}$.

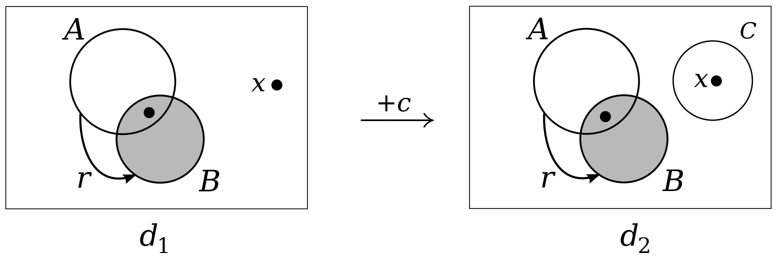


Figure 3.24: Transforming a diagram by adding a contour.

Formal description. Let d_1 be a unitary diagram and let c be a contour that is not in d_1. Let Z_{in} and Z_{out} be subsets of $Z(d_1)$ such that $Z_{in} \cup Z_{out} = Z(d_1)$. Let S_{in} and S_{out} be a two-way partition of $S(d_1)$ such that

1. for all x in S_{in}, $\eta_{d_1}(x) \in Z_{in}$ and

2. for all y in S_{out}, $\eta_{d_1}(y) \in Z_{out}$.

Let d_2 be a unitary diagram where

1. $C(d_2) = C(d_1) \cup \{c\}$,
2. $Z(d_2) = \mathit{in} \cup \mathit{out} \cup \mathit{split}$, where
   (a) $\mathit{in} = \{(a \cup \{c\}, b) : (a,b) \in Z_{in} - Z_{out}\}$,
   (b) $\mathit{out} = \{(a, b \cup \{c\}) : (a,b) \in Z_{out} - Z_{in}\}$,
   (c) $\mathit{split} = \{(a \cup \{c\}, b), (a, b \cup \{c\}) : (a,b) \in Z_{in} \cap Z_{out}\}$,
3. $Z^*(d_2) = \mathit{in}^* \cup \mathit{out}^* \cup \mathit{split}^*$, where
   (a) $\mathit{in}^* = \{(a \cup \{c\}, b) : (a,b) \in (Z_{in} - Z_{out}) \cap Z^*(d_1)\}$,
   (b) $\mathit{out}^* = \{(a, b \cup \{c\}) : (a,b) \in (Z_{out} - Z_{in}) \cap Z^*(d_1)\}$,
   (c) $\mathit{split}^* = \{(a \cup \{c\}, b), (a, b \cup \{c\}) : (a,b) \in (Z_{in} \cap Z_{out}) \cap Z^*(d_1)\}$,
4. $S(d_2) = S(d_1)$,
5. for each $x \in S(d_2)$, where $\eta_{d_1}(x) = z$,
   (a) $x \in S_{in} \Rightarrow \eta_{d_2}(x) = (\mathit{in}(z) \cup \{c\}, \mathit{out}(z))$,
   (b) $x \in S_{out} \Rightarrow \eta_{d_2}(x) = (\mathit{in}(z), \mathit{out}(z) \cup \{c\})$,
6. $\tau_{d_2} = \tau_{d_1}$,
7. $A(d_2) = A(d_1)$.

Then d_1 can be transformed into d_2 by using the **add contour** transformation to add c, given a choice of Z_{in}, Z_{out}, S_{in} and S_{out}, denoted $d_1 \xrightarrow{+P} d_2$, where $P = (c, Z_{in}, Z_{out}, S_{in}, S_{out})$.

3.2.3 Completeness of the unitary transformations

We now show that we have a complete set of unitary transformations, in the sense that any generalized unitary diagram can be transformed into any other. We show this by describing a naive method in which, to transform a unitary diagram d_1 into a second unitary diagram d_2, we remove all syntax from d_1 then add syntax until d_1 has all the same syntactic elements as d_2. There will often be faster and more elegant ways to transform one unitary diagram into another, but this method suffices to show the completeness of the set of unitary transformations. Transforming the unitary diagrams relies on choosing the right order in which to apply the transformations, depending on their preconditions of syntactic well-formedness; for instance, before using the remove spider transformation to remove a spider x which is the source of an arrow a in d', we must first use the remove arrow transformation to remove a from d_1.

Theorem 3.2.1: Let d_1 and d_2 be generalized unitary constraint diagrams where $d_1 \neq \perp$ and $d_2 \neq \perp$. Then there is a sequence of transformations which can be used to transform d_1 into d_2.

Proof. Assume that $A(d_1)$ is not empty and that d_1 has a set of arrows $\{a_1 \dots a_n\}$. We begin by applying the remove arrow transformation to remove a_1 from d_1, obtaining diagram e_1. We continue to remove the rest of the arrows, obtaining diagram e_n. None of the spiders in e_n are arrow ends, and so its spiders can now be removed by repeated use of the remove spider transformation, obtaining diagram f_m, where m is the number of spiders in d_1.

We next remove the shading from each zone in f_m, obtaining diagram g_o, where o is the number of shaded zones in f_m, and then remove the contours of g_o in a similar way, obtaining diagram h_p, where p is the number of contours in g_o. The diagram h_p is empty of syntax other than for the single zone $(\emptyset, \emptyset)$.

Next, we use the add contour transformation to add the contours in $C(d_2)$ to h_p. Recall that the add contour transformation is parametrised with two sets of zones, Z_{in} and Z_{out}, which are the sets of zones of the original diagram which will be inside and outside the new contour, and similarly with the sets of spiders which are inside and outside the new contour, S_{in} and S_{out}. The diagram h_p contains no spiders, so $S_{in} = S_{out} = \emptyset$. We will add each contour so that it splits every zone in the diagram to which it is added. Thus, to add the first contour, say c_1, we set $Z_{in} = Z_{out} = Z(h_p)$ and add c_1 to h_p obtaining diagram i_1. For each subsequent contour, c_i, we set $Z_{in} = Z_{out} = Z(i_{i-1})$. We follow this process to obtain diagram i_q, where q is the number of contours in d_2. Note that i_q is a Venn diagram.

Next we use the remove zone transformation to remove the zones in $Z(i_q) - Z(d_2)$, producing the unitary diagram j which has the same contours and zones as d_2. Next, we use the add shading transformation to the zones $Z^*(d_2)$ in j, producing the unitary diagram k. We continue in a similar manner using the add spider, add spider with tie and add arrow transformations in that order to produce a diagram, l, with the same contours, zones, shading, spiders, ties and arrows as d_2. Since d_2 is finite, the process will terminate and thus the set of unitary transformations is complete.

□

3.3 Extending the decision procedure to the unitary fragment

Our goal is to use the decision procedure for γ-diagrams described in section 3.1.3 in larger fragments of the unitary GCD system. In this section we will show that we can do this by converting a unitary diagram, d, to a set of unitary γ-diagrams, Γ, and that d is satisfiable if and only if one or more element of Γ is satisfiable. The first step in the process is to remove spiders from d until d contains no spiders which are joined by ties. This is necessary because γ-diagrams are defined so that they contain no ties (other than the ties from each spider to itself, which are not depicted in diagrams). This restriction in the definition of γ-diagrams is made for convenience. In Figure 3.25, we can remove spiders from d_1 until no ties between spiders are left, obtaining d_2. This has the effect of reducing the cardinality of each equivalence class of τ_{d_1} to one. We say that d_2 is in *singleton-τ form*, and is an *associated* singleton-τ form of d_1. Because ties between spiders represent equality, it is easy to see that d_1 has an equivalent meaning to d_2.

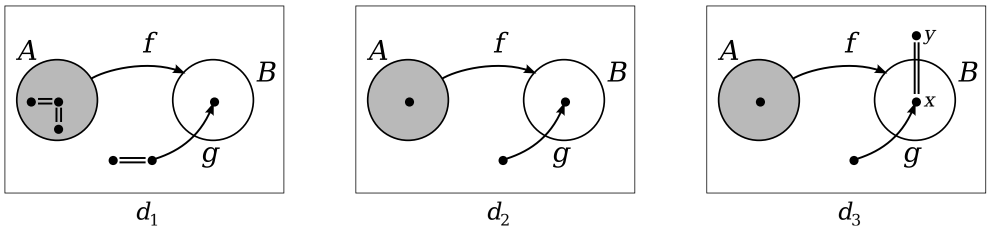


Figure 3.25: Removing spiders to obtain singleton-τ form.

Definition 3.3.1. Let d be a generalized unitary α-diagram. If $\tau_d = \{(x, x) : x \in S(d)\}$, we say that d is in **singleton-τ form**.

In Figure 3.25, diagram d_3, two spiders with different habitats are joined by a tie. This diagram is inconsistent. For the spiders distinctness condition to be true, the spiders labelled x and y must represent the same element. However, for the spiders' habitat condition to be true, x and y must represent distinct elements. We will show in section 3.3.3 that such diagrams are unsatisfiable.

After converting a unitary diagram, d_1, to an equivalent diagram, d_2, in singleton-τ form, the next step is to transform d_2 to an equivalent diagram, d_3, in which all arrows are sourced on contours; we say that d_3 is in *contour-source form.*

Definition 3.3.2. Let d be a generalized unitary α-diagram. If each arrow $(l, s, t) \in A(d)$ is sourced on a contour, that is $s \in C(d)$, we say that d is in **contour-source form**.

To reiterate, our goal in making this series of transformations is to produce a set of γ-diagrams. We transform diagrams in singleton-τ form to contour-source form rather to sets of β-diagrams because it is not possible, in general, to convert a β-diagram, d, to a γ-diagram, d', in one step. This is because replacing the arrows sourced on spiders in d may result in zones which are unshaded and contain no spiders. In Figure 3.26, d_1 is a β-diagram in singleton-τ form in which an arrow is sourced on a spider. Diagram d_2 is a copy of d_1 in which the arrow sourced on the spider x has been replaced by an arrow sourced on a fully shaded derived contour that contains only x. However, d_2 is not a β-diagram, and thus not a γ-diagram, since the zone outside all contours is unshaded and contains no spiders.

To achieve the transformation to singleton-τ and contour-source forms, we formulate a series of sound inference rules. When used in sequence, these rules allow us to remove spiders until there are no ties remaining in a diagram, and to replace each arrow sourced on a spider, x, with an arrow sourced on a derived contour which contains just x. In section 3.3.1 we define the rules, and in section 3.3.2 we show them to be sound. In section 3.3.3 we show that every α-diagram, d_1, has an associated diagram, d_2, where d_2 is in singleton-τ form and is semantically equivalent to d_1. Furthermore, we show that every diagram, d_3,

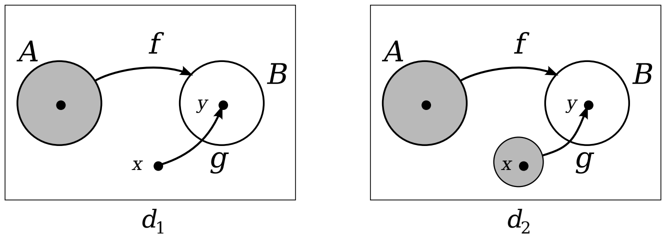


Figure 3.26: β-diagrams and contour-source form.

in singleton-τ form, has an associated diagram in contour-source form, d_4, where d_3 is semantically equivalent to d_4, and we use these facts to extend the decision procedure for γ-diagrams to the unitary existential fragment.

3.3.1 Inference rules

The first rule we require is one which removes a spider which is joined by a tie to another in a unitary diagram. In Figure 3.27 we can remove the spider labelled y from diagram d_1 to obtain d_2. Note that the arrow sourced on y in d_1 is replaced in d_2 by an arrow with the same label and target and which is sourced on one of the spiders equal to y. The rule is defined to replace all arrows sourced on or targeting the spider to be removed. This rule can be used repeatedly to produce a diagram in singleton-τ form.

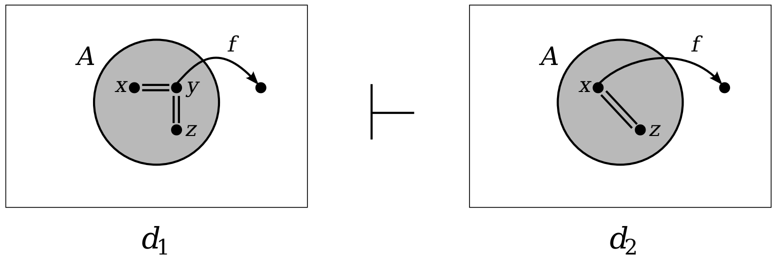


Figure 3.27: An application of *remove equal spider.*

Rule 1: Remove equal spider. Let d_1 be a generalized unitary diagram and let x and y be spiders in $S(d_1)$ which satisfy the following:

1. $(x, y) \in \tau_{d_1}$,

2. $\eta_{d_1}(x) = \eta_{d_1}(y)$, and
3. $x \neq y$.

Let d_2 be the generalized unitary diagram which satisfies the following:

1. $Z(d_2) = Z(d_1)$,
2. $Z^*(d_2) = Z^*(d_1)$,
3. $S(d_2) = S(d_1) - \{x\}$,
4. $\eta_{d_2} = \eta_{d_1}|_{S(d_2)}$, and
5. $A(d_2) = \{f(l,s,t) : (l,s,t) \in A(d_1)\}$, where $f : A(d_1) \to A(d_2)$ is a bijection defined as follows:

$$f(l,s,t) = \begin{cases} (l,s,t) & \text{if } s \neq x \wedge t \neq x \\ (l,y,t) & \text{if } s = x \wedge t \neq x \\ (l,s,y) & \text{if } s \neq x \wedge t = x \\ (l,y,y) & \text{if } s = x \wedge t = x. \end{cases}$$

Then d_1 can be replaced by d_2 and vice versa.

Next, we consider the process of ensuring that all arrows are sourced on contours. To transform d_1 in Figure 3.26 to d_2, we need to transform the initial diagram four times:

1. by adding a new derived contour containing just the spider x,
2. by shading the zone inside the new derived contour,
3. by adding an arrow sourced on the new contour and with the same label and target as (g,x,y) and, finally,
4. by removing the arrow (g,x,y).

We combine the first two steps to define a rule which adds a fully shaded contour, and we combine steps three and four to define a rule which replaces arrows. Both rules are defined in the context of initial diagrams in singleton-τ form.

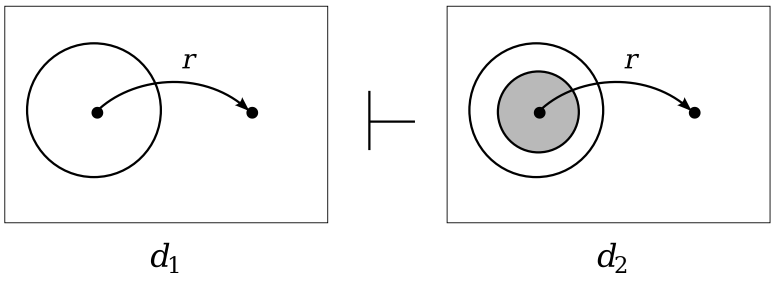


Figure 3.28: An application of *add contour over spider*.

Rule 2: Add contour over spider. Let d_1 be a generalized unitary diagram in singleton-τ form and let x be a spider in $S(d_1)$. Let dc be a derived contour such that $dc \notin C(d_1)$. Let d_1' be the diagram obtained by using transformation 12, add contour, to add dc to d_1: $d_1 \xrightarrow{+P} d_1'$, where $P = (Z_{in}, Z_{out}, S_{in}, Z_{out})$ satisfies the following:

1. $Z_{in} = \eta_{d_1}(x)$,
2. $Z_{out} = Z(d_1)$,
3. $S_{in} = \{x\}$,
4. $S_{out} = S(d_1) - \{x\}$.

Let d_2 be the diagram obtained by adding shading to the habitat of the spider x in d_1': $d_1' \xrightarrow{+(\eta_{d_1'}(x))^*} d_2$. Then d_1 can be replaced by d_2 and vice versa.

Next, we need a rule that replaces an arrow, (l, x, t), where the source, x, is the only spider within a fully shaded contour, c, with the arrow (l, c, t). In Figure 3.29 the spiders are given the labels x and y for convenience. In diagram d_1, x is the source of an arrow, (r, x, y). The spider x is the only spider inhabiting a zone inside the contour A, and all zones inside A are shaded. This allows us to infer that we can add an arrow sourced on A, with the same label and target as (r, x, y), as shown in diagram d_2.

Rule 3: Replace arrow. Let d_1 be a generalized unitary diagram in singleton-τ form and let (l, x, t) be an arrow in $A(d_1)$, where x is the only spider within a fully shaded contour c. Let d_1' be the diagram obtained using transformation 7,

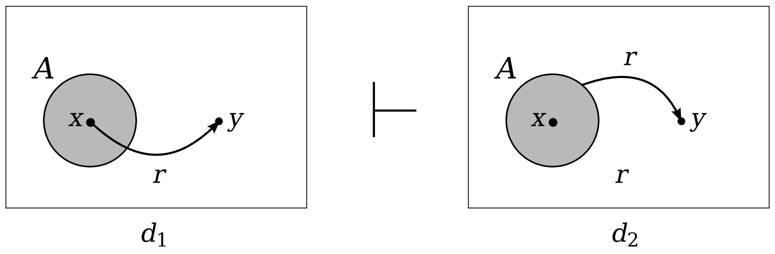


Figure 3.29: An application of *replace arrow*.

add arrow, to add (l,c,t) to d_1: $d_1 \xrightarrow{+(l,c,t)} d_1'$. Let d_2 be the diagram obtained by removing (l,x,t) from d_1': $d_1' \xrightarrow{-(l,x,t)} d_2$. Then we can replace d_1 with d_2 and vice versa.

3.3.2 Validity of the inference rules

We will show that the rules defined in the previous section are sound. Rule 1, remove equal spider, is justified by the fact that ties represent equality.

Theorem 3.3.1: Rule 1, remove equal spider, is sound. Let d_1 be a generalized unitary diagram and let x and y be spiders in $S(d_1)$ which satisfy the following:

1. $(x,y) \in \tau_{d_1}$,
2. $\eta_{d_1}(x) = \eta_{d_1}(y)$, and
3. $x \neq y$.

Let d_2 be the diagram obtained by using the remove equal spider rule to remove x from d_1. Then $d_1 \equiv_{\vDash} d_2$.

Proof. First we show that $d_1 \vDash d_2$. Let $I = (U, \Psi, \Phi)$ be a model for d_1 with valid extension $I' = (U, \Psi', \Phi)$. The plane tiling, shaded zones, spiders habitat and distinctness conditions for d_1 imply those for d_2. For each arrow, (l,s,t), in $A(d_1)$, then if $s \neq x$ and $t \neq x$, (l,s,t) is in d_2. Otherwise, if $s = x$ or $t = x$ or $s = t = x$, then (l,s,t) is replaced in d_2 by an arrow which is either sourced on y, or targets y, or is sourced on and targets y respectively. By the spiders' distinctness condition for d_1, $\Psi'(x) = \Psi'(y)$, and so we can see that arrows condition is true for d_2 if it is true for d_1, and $d_1 \vDash d_2$. To show that $d_2 \vDash d_1$, let $I = (U, \Psi, \Phi)$

be a model for d_2 with valid extension $I' = (U, \Psi', \Phi)$. Let $I'' = (U, \Psi'', \Phi)$ be an extension of I' such that $\Psi''(x) = \Psi'(y)$. Then it follows that I'' is valid for d_2, so $d_2 \vDash d_1$ and $d_1 \equiv_{\vDash} d_2$ as required. $\square$

Next, we show that rule 2, add contour over spider, is sound. Informally, this rule is justified by the fact that we can add arbitrary derived contours to a diagram.

Theorem 3.3.2: Rule 2, add contour over spider, is sound. Let d_1 be a generalized unitary diagram in singleton-τ form and let x be a spider in $S(d_1)$. Let dc be a derived contour such that $dc \notin C(d_1)$. Let d_2 be the diagram obtained by adding dc to d_1 using the add contour over spider rule. Then $d_1 \equiv_{\vDash} d_2$.

Proof. To show that $d_1 \vDash d_2$, let (U, Ψ, Φ) be a model for d_1, and let Ψ'_1 be a valid extension of Ψ for d_1. Let $\Psi'_2 : \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S} \cup \mathcal{Z}$ be an extension of Ψ'_1 which satisfies

1. $\Psi'_2(y) = \Psi'_1(y)$ for each $y \in S(d_2)$,
2. $\Psi'_2(c) = \Psi'_1(c)$ for each $c \in C(d_2) - \{dc\}$, and
3. $\Psi'_2(dc) = \Psi'_1(x)$.

We will show that Ψ'_2 is a valid extension of (U, Ψ, Φ) for d_2. We show that the spiders habitat condition holds first. For each spider $y \in S(d_2)$ we need to show that $\Psi'_2(y) \subseteq \Psi'_2(\eta_{d_2}(y))$. We know that $\Psi'_2(y) = \Psi'_1(y)$. Let $(in, out) = \eta_{d_1}(y)$. Then we can deduce the following:

$$\begin{aligned}
\Psi'_2(y) &\subseteq \Psi'_1(in, out) && \text{(habitats condition for } d_1\text{)} \\
&= \bigcap_{c \in in} \Psi'_1(c) \cap \bigcap_{c \in out} (U - \Psi'_1(c)) && \text{(definition of } \Psi'_1\text{)} \\
&= \bigcap_{c \in in} \Psi'_2(c) \cap \bigcap_{c \in out} (U - \Psi'_2(c)) && \text{(definition of } \Psi'_2.\text{)} \qquad (3.13)
\end{aligned}$$

If $y \neq x$ then $\eta_{d_2}(y) = (in, out \cup \{dc\})$ and by the spiders' distinctness condition for d_1 we know

$$\Psi'_1(y) \cap \Psi'_1(x) = \emptyset.$$

Since $\Psi'_1(y) = \Psi'_2(y)$ and $\Psi'_1(x) = \Psi'_2(x) = \Psi'_2(dc)$ it follows that

$$\Psi'_2(y) \cap \Psi'_2(dc) = \emptyset.$$

Therefore, by (3.13),

$$\begin{aligned}\Psi'_2(y) &\subseteq \bigcap_{c \in in} \Psi'_2(c) \cap \bigcap_{c \in out} (U - \Psi'_2(c)) \cap (U - \Psi'_2(dc)) \\ &\subseteq \Psi'_2(in, out \cup \{dc\}) \\ &\subseteq \Psi'_2(\eta_{d_2}(y)).\end{aligned}$$

We now consider the case when $y = x$. Then $\eta_{d_2}(x) = (in \cup \{dc\}, out)$. Since $\Psi'_2(x) = \Psi'_2(dc)$, it follows from (3.13) that

$$\Psi'_2(x) \subseteq \bigcap_{c \in in} \Psi'_2(c) \cap \bigcap_{c \in out} \Psi'_2(c) \cap \Psi'_2(dc).$$

Therefore, $\Psi'_2(x) \subseteq \Psi'_2(in \cup \{dc\}, out)$. The spiders habitat condition therefore holds for d_2. We next show that the plane tiling condition holds, that is,

$$\bigcup_{z \in Z(d_2)} \Psi'_2(z) = U.$$

We know that, in d_1,

$$\bigcup_{z \in Z(d_1)} \Psi'_1(z) = U.$$

Let (a, b) be a zone in $Z(d_1)$; then there is a zone $z = (a, b \cup \{dc\})$ in $Z(d_2)$. Now,

$$\begin{aligned}\Psi'_2(z) &= \bigcap_{c \in a} \Psi'_2(c) \cap \bigcap_{c \in b \cup \{dc\}} (U - \Psi'_2(c)) \\ &= \bigcap_{c \in a} \Psi'_1(c) \cap \bigcap_{c \in b} (U - \Psi'_1(c)) \cap (U - \Psi'_2(dc)) \\ &= \Psi'_1(a, b) \cap (U - \Psi'_2(dc)). \qquad (3.14)\end{aligned}$$

The zone $\eta_{d_1}(x) = (in, out)$ in $Z(d_1)$ is replaced in d_2 by two zones, $(in \cup$

$\{dc\}, out)$ and $(in, out \cup \{dc\})$. Let

$$Z_2 = Z(d_2) - \{(in \cup \{dc\}, out), (in, out \cup \{dc\})\}.$$

Then we know that

$$\begin{aligned}\bigcup_{z \in Z_2} \Psi'_2(z) &= \bigcup_{z \in Z(d_1) - \{(in, out)\}} (\Psi'_1(z) \cap (U - \Psi'_2(dc))) \qquad \text{(by (3.14))}\\ &= \bigcup_{z \in Z(d_1) - \{(in, out)\}} \Psi'_1(z) \cap (U - \Psi'_2(dc))\end{aligned}$$

and therefore, by the plane tiling condition for d_1,

$$\bigcup_{z \in Z_2} \Psi'_2(z) = (U - \Psi'_1(in, out)) \cap (U - \Psi'_2(dc)). \tag{3.15}$$

We know that $\Psi'_2(dc) = \Psi'_1(x) \subseteq \Psi'_1(in, out)$ by the habitat condition for d_1, and therefore $\Psi'_2(dc) \subseteq \Psi'_1(in, out)$ and (3.15) becomes

$$\bigcup_{z \in Z_2} \Psi'_2(z) = U - \Psi'_1(in, out).$$

By corollary 2.3.1,

$$\Psi'_1(in, out) = \Psi'_2(in \cup \{dc\}, out) \cup \Psi'_2(in, out \cup \{dc\}).$$

Therefore,

$$\bigcup_{z \in Z(d_2)} \Psi'_2(z) = U.$$

We can see that as the spiders' distinctness and arrows conditions hold for d_1 they will also hold for d_2. Hence, $d_1 \vDash d_2$.

To show the reverse, that $d_2 \vDash d_1$, we need to reason in a similar way about the habitat of the spiders and the zone sets of d_1 and d_2. We omit this part of the proof as it readily extends from existing results such as the delete curve rule in [26]. Hence, $d_1 \equiv_{\vDash} d_2$. □

The next rule, replace arrow, is justified by the fact that arrows sourced on spiders are considered, semantically, to be sourced on singleton sets. Therefore if an arrow a is sourced on a spider which is the only spider within a fully shaded contour c, we can add an arrow with the same label and target as a which is sourced on c.

Theorem 3.3.3: Rule 3, replace arrow, is sound. Let d_1 be a generalized unitary diagram where there is an arrow (l, x, t) in $A(d_1)$, where x is the only spider within a fully shaded contour c. Let d_2 be the diagram obtained by using the replace arrow rule to replace (l, x, t) by (l, c, t) in d_1. Then $d_1 \equiv_{\models} d_2$.

Proof. Let (U, Ψ, Φ) be a model for d_1 and let Ψ' be a valid extension for d_1. We show that Ψ' is a valid extension for d_2. Let d_1' be the diagram obtained using transformation 7, add arrow, to add (l, c, t) to d_1: $d_1 \xrightarrow{+(l,c,t)} d_1'$. By the arrows condition for d_1,

$$\Psi'(x).\Phi(l) = \Psi'(t). \tag{3.16}$$

We know that c contains a single zone which is the habitat of the spider x. By the shaded zones and spiders habitat condition for d_1,

$$\Psi'(Z(c, d_1)) = \Psi'(S(c, d_1)) = \Psi'(x). \tag{3.17}$$

By the definition of Ψ' for zones, $\Psi'(c) = \Psi'(Z(c, d_1))$ so (3.17) becomes $\Psi'(c) = \Psi'(x)$ and we can rewrite (3.16) as

$$\Psi'(c).\Phi(l) = \Psi'(t).$$

Hence the arrows condition holds for d_1'. The other conditions are not affected by the addition of (l, c, t) and so $d_1 \models d_1'$. Diagram d_2 is the diagram obtained by removing (l, x, t) from d_1'. Since $A(d_2) \subset A(d_1')$, the arrows condition for d_1' implies that of d_2. The other conditions are unaffected, so $d_1' \models d_2$ and hence $d_1 \models d_2$. We can show that $d_2 \models d_1$ by a similar argument and so $d_1 \equiv_{\models} d_2$ as required. □

3.3.3 Transforming α-diagrams to sets of γ-diagrams

We now show that we can use the inference rules defined and proved sound in the preceding sections to transform an α-diagram into singleton-τ form, then into contour-source form and, finally, into a set of γ-diagrams. First, we show that every unitary diagram, d, has an associated unitary diagram in which no ties are present. We call this diagram an *associated singleton-τ form* of d. To define associated singleton-τ forms we make use of the notion of equivalence class representatives.

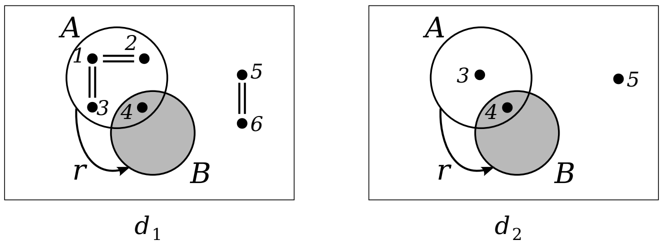


Figure 3.30: Equivalence class representatives of τ.

In Figure 3.30, spiders are numbered for convenience. Diagram d_2 is an associated singleton-τ form of diagram d_1. In d_1, the relation τ_{d_1} relates each spider to itself and also, because of the ties present in d_1, relates 1, 2 and 3 to each other, and relates 5 and 6 to each other. The relation τ_{d_1} has the following equivalence classes:

1. $\{1, 2, 3\}$,
2. $\{4\}$,
3. $\{5, 6\}$.

A set of class representatives of τ_{d_1} is formed by choosing one element from each equivalence class; thus, $\{1, 4, 5\}$ and $\{3, 4, 6\}$ are sets of class representatives of τ_{d_1}. In Figure 3.30, the set of spiders present in diagram d_2 is a set of class representatives of τ_{d_1}. Since spiders joined by ties are equal, semantically, it does not matter which set of class representatives is used.

Definition 3.3.3. Let d_1 be a generalized unitary α-diagram. Define an **associated singleton-τ form** of d_1 as the generalized unitary diagram, d_2, which satisfies the following:

1. $Z(d_1) = Z(d_2)$,
2. $Z^*(d_1) = Z^*(d_2)$,

3. the set of spiders of d_2 is a set of class representatives of τ_{d_1},
4. $\eta_{d_2} \subseteq \eta_{d_1}$,
5. there is a bijection, $f : A(d_1) \to A(d_2)$, which satisfies the following:

$$f(l,s,t) = \begin{cases} (l,s,t) & \text{if } s,t \in S(d_2) \cup C(d_2) \\ (l,s,t') & \text{if } s \in S(d_2) \cup C(d_2) \wedge t \in [t']_{d_1} \\ (l,s',t) & \text{if } t \in S(d_2) \cup C(d_2) \wedge s \in [s']_{d_1} \\ (l,s',t') & \text{if } s \in [s']_{d_1} \wedge t \in [t']_{d_1} \end{cases}$$

Lemma 3.3.1. Let d_1 be a unitary α-diagram in which all pairs of spiders, x and y, which are joined by ties inhabit the same zone: $(x,y) \in \tau_{d_1} \Rightarrow \eta_{d_1}(x) = \eta_{d_1}(y)$. Let d_2 be an associated singleton-τ form of d_1. Then $d_1 \equiv_{\models} d_2$.

Proof. Diagram d_2 can be obtained from d_1 by the repeated use of the remove equal spider rule. Equally, diagram d_1 can be obtained from d_2 by using the same rule in reverse. By theorem 3.3.1, which proves the soundness of that rule, $d_1 \equiv_{\models} d_2$. □

Lemma 3.3.2. Let d be a generalized unitary α-diagram in which all pairs of spiders, x and y, which are joined by ties inhabit the same zone: $(x,y) \in \tau_{d_1} \Rightarrow \eta_{d_1}(x) = \eta_{d_1}(y)$. Then we can use a finite series of inference rule applications to produce a diagram, d_2, where d_2 is an associated singleton-τ form of d_1 and $d_1 \equiv_{\models} d_2$.

Proof. Since $\eta_{d_1}(x) = \eta_{d_1}(y)$ for each $(x,y) \in \tau_{d_1}$ where $x \neq y$, then we use rule 1, remove equal spider, to remove each x from d_1, resulting in a diagram, d_2, which is an associated singleton-τ form of d_1. Since d_1 has finite spiders, this process is terminating. By lemma 3.3.1, $d_1 \equiv_{\models} d_2$. □

The previous two results depend on spiders which are joined by ties having the same habitat. Next we show that, if this is not true, the diagram containing the spiders is unsatisfiable. We use this fact to discard inconsistent diagrams when transforming to singleton-τ form.

Lemma 3.3.3. Let d be a unitary α-diagram which contains two spiders, x and y, where $(x,y) \in \tau_d$ and $\eta_d(x) \neq \eta_d(y)$. Then d is unsatisfiable.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. By the spiders' distinctness condition, $\Psi'(x) = \Psi'(y)$. We know that $\eta_d(x) \neq \eta_d(y)$ and, since distinct zones represent distinct subsets of U,

$$\Psi'(\eta_d(x)) \cap \Psi'(\eta_d(y)) = \emptyset.$$

By the spiders habitat condition, $\Psi'(x) \subseteq \Psi'(\eta_d(x))$ and $\Psi(y) \subseteq \Psi'(\eta_d(y))$. Thus, $\Psi'(x) \neq \Psi'(y)$, a contradiction which shows that I' is invalid for d and d is unsatisfiable. □

We can now move on to transforming unitary diagrams in singleton-τ form to contour-source form.

Definition 3.3.4. Let d_1 be a generalized unitary α-diagram in singleton-τ form. Define an **associated contour-source form** of d_1 as the generalized unitary diagram, d_2 where, for each arrow $(l, x, t) \in A(d_1)$ where $x \in S(d_1)$, (l, x, t) is replaced in $A(d_2)$ by an arrow (l, dc, t), where dc is the contour that results from using rule 2, add contour over spider, to add a new derived contour to d_1 with regard to x.

We will now show that if d_1 is a unitary α-diagram in singleton-τ form then we can obtain a diagram, d_2, where d_2 is an associated contour-source form of d_1 and $d_1 \equiv_{\vDash} d_2$. We do this by using a series of inference rules. In Figure 3.31, the only arrow in d_1, (r, x, t), has a spider as its source. In d_2 we use the add contour over spider rule to add a new, fully shaded, derived contour containing a single zone which is the new habitat of x. In d_3 we use the replace arrow rule to add an arrow with the derived contour as its source, label r and target t, and to simultaneously remove the arrow (r, x, t), obtaining an associated contour-source form of d_1.

Lemma 3.3.4. Let d_1 $(\neq \perp)$ be a generalized unitary α-diagram in singleton-τ form. Then there is a finite series of inference rules that can be used to obtain a diagram, d_2, where d_2 is an associated contour-source form of d_1 and $d_1 \equiv_{\vDash} d_2$.

Proof. First, we show that $d_1 \vDash d_2$. Let $\{a_1 \dots a_n\}$ be the arrows in d_1 which are sourced on spiders. For each $(l, x_i, t_i) \in \{a_1 \dots a_n\}$, and each corresponding derived contour dc_i, we need to show that d_1 entails the diagram obtained by

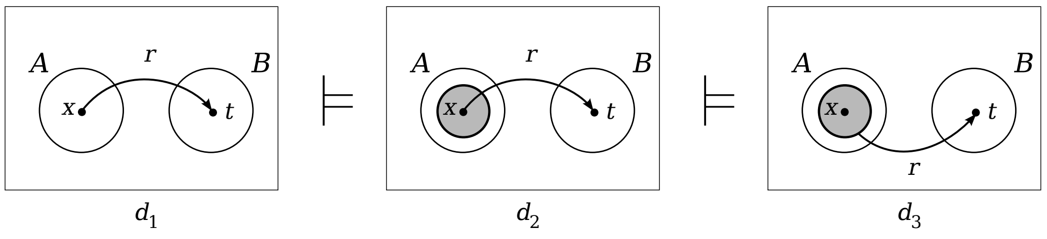


Figure 3.31: Obtaining contour-source form.

adding both dc_i and (l, dc_i, t_i), then removing (l, x_i, t_i) from d, where the resulting diagram satisfies the post conditions of the add contour over spider rule with regard to the spider x_i.

Let (l, x, t) be an arrow sourced on a spider in d_1, and let e_1 be the diagram obtained by using rule 2, add contour over spider, to add a new derived contour, dc, over x in d_1. Then $d_1 \equiv_{\vDash} e_1$ by theorem 3.3.2. By the post conditions of the add contour rule, $S(dc, e_1) = \{x\}$, $Z(dc, e_1) = \{(in \cup \{dc\}, out)\}$, where $(in, out) = \eta_{d_1}(x)$ and $Z(dc, e_1) \subseteq Z^*(e_1)$. The syntactic conditions therefore exist for the application of rule 3, replace arrow, to replace (l, x, t) by (l, dc, t); let e_2 be the diagram obtained by this operation. Then $e_1 \vDash e_2$ by theorem 3.3.3. By the definition of contour-source form, d_2 is the diagram obtained by repeating this process for all arrows in $\{a_1 \ldots a_n\}$, and it follows that $d_1 \vDash d_2$. Since d_1 has finite arrows, the process terminates.

We can show that $d_2 \vDash d_1$ using similar reasoning. The process requires a series of applications of rule 3, replace arrow, and rule 2, add contour over spider (used in reverse to remove contours). Thus, $d_2 \vDash d_1$ and $d_1 \equiv_{\vDash} d_2$ as required. □

We can now show that each unitary α-diagram in single-τ and contour-source forms, d_1, entails the disjunction of a set of γ-diagrams which we call the γ *components* of d_1. It follows that, given a unitary α-diagram d_2, we can use the method given in section 3.3 to find an equivalent diagram, d_2, where d_2 is in singleton-τ and contour-source forms, then use the decision procedure of section 3.1 to say that d_2 is satisfiable if one or more of its γ components is satisfiable.

In Figure 3.32, d_1 is an α-diagram in singleton-τ and contour-source forms which contains a zone, $(\{A\}, \{B\})$, which is unshaded and contains no spiders. All other zones in d_1 are shaded or contain a spider, or both. Semantically, the

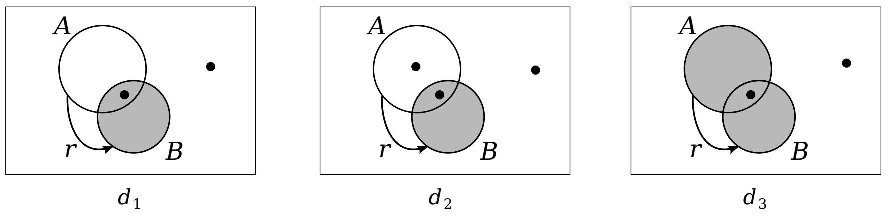


Figure 3.32: An α-diagram and its γ components.

absence of shading in a zone means that the zone contains at least as many elements as are depicted by spiders. It follows that d_1 semantically entails either d_2, which is a copy of d_1 in which $(\{A\}, \{B\})$ contains an extra spider, or d_3, in which that zone is shaded.

Definition 3.3.5. Let d_1 be a generalized unitary α-diagram in singleton-τ and contour-source forms, and let d_2 be a generalized unitary γ-diagram. We say that d_2 is a γ **component** of d_1 if the following conditions hold:

1. the diagrams d_1 and d_2 have the same set of zones,
2. for every shaded zone, z, in d_1, z is shaded in d_2 and $S(z, d_2) = S(z, d_1)$,
3. for every unshaded zone that contains spiders in d_1, z, z is unshaded in d_2 and $S(z, d_2) = S(z, d_1)$,
4. every unshaded zone that contains no spiders in d_1 is either unshaded and contains a single spider in d_2, or is shaded and contains no spiders in d_2, and
5. the diagrams d_1 and d_2 have the same set of arrows.

Theorem 3.3.4: Let d ($\neq \bot$) be a generalized unitary α-diagram in singleton-τ and contour-source forms. Then d is satisfiable if and only if one or more of its γ components is satisfiable.

Proof. We first show that, if d is satisfiable, one of its γ components is satisfiable. Let (U, Ψ, Φ) be a model for d and let Ψ'_1 be a valid extension for d. Choose a γ component of d, say g, where for each zone $z \in Z(d)$ which is unshaded and contains no spiders, if $\Psi'_1(z) = \emptyset$ then z is shaded in g, and if $\Psi'_1(z) \neq \emptyset$ then z is unshaded and contains a single spider in g.

Let $\Psi'_2 : \mathcal{GC} \cup \mathcal{DC} \cup \mathcal{S}$ be an extension of Ψ defined by $\Psi'_2(c) = \Psi'_1(c)$ for each $c \in DC(g)$ and

$$\Psi'_2(x) = \begin{cases} \Psi'_1(x) & \text{if } x \in S(d) \\ \{e\} \text{ for some } e \in \Psi'_1(\eta_d(x)) & \text{if } x \in S(g) - S(d). \end{cases}$$

In the second case of $\Psi'_2(x)$, where $x \in S(g) - S(d)$, we know there are elements available in $\Psi'_1(\eta_d(x))$, as the only spiders in g but not in d are those in a zone, z, where $\Psi'_1(z) \neq \emptyset$. We will show that Ψ'_2 is a valid extension for g and therefore (U, Ψ, Φ) is a model for g.

The plane tiling condition holds trivially, as $Z(d) = Z(g)$ and the plane tiling condition holds for d_1. The spiders' distinctness condition holds for g since every spider in $S(g) - S(d)$ is the only spider in its habitat and the spiders' distinctness condition holds for d. The spiders habitat condition also holds trivially. To show the shaded zones condition holds for g, we will show that, for every shaded zone $z \in Z^*(g)$, $\Psi'_1(z) = \Psi'_2(z)$. By the shaded zones condition for d,

$$\bigcup_{z \in Z^*(d)} \Psi'_1(z) = \bigcup_{x \in S(z,d)} \Psi'_1(x).$$

If $z \in Z^*(g) - Z^*(d)$, $\Psi'_1(z) = \emptyset$, by our choice of g. Hence, $\Psi'_2(z) = \Psi'_1(z) = \emptyset$. The shaded zones condition therefore holds for g. The arrows condition is unaffected, and so (U, Ψ, Φ) satisfies g. Therefore if d is satisfiable then so is one of its γ components.

To show the reverse, let g be a γ component of d, let (U, Ψ, Φ) be a model for g and let Ψ'_2 be a valid extension for g. The shaded zones condition for g implies that of d, since $Z^*(d) \subset Z^*(g)$.

For each zone, z, which is unshaded and contains no spiders in d, z is either shaded and contains no spiders in g, or is unshaded and contains a single spider in g. In all other zones, z, (i.e. z is shaded in d, or contain spiders in d, or both), $S(z, g) = S(z, d)$. Thus, the spiders' habitat and distinctness conditions for g imply those for d. It must be then that (U, Ψ, Φ) is a model for d, and $g \vDash d$. Hence, d is satisfiable if and only if one of its γ components is satisfiable. $\square$

Theorem 3.3.5: The unitary α fragment is decidable.

Proof. We determine the satisfiability of a unitary α-diagram, d, as follows. If there exists a pair of spiders, x and y, where $(x, y) \in \tau_d$ and $\eta_d(x) \neq \eta_d(y)$ then, by lemma 3.3.3, d is unsatisfiable. Assume that this is not the case. Then, by lemma 3.3.2, we can produce a diagram, d', where d' is an associated singleton-τ form of d and $d \equiv_{\models} d'$. By lemma 3.3.4, we can construct d'', an associated contour-source form of d', in finite steps. Finally, we construct the set of γ components of d''; then, by lemma 3.3.4 and theorem 3.3.4, d is satisfiable if and only if one or more of the γ components of d' is satisfiable. Hence, the unitary α fragment is decidable. $\square$

The proof of theorem 3.3.5 encapsulates the decision procedure for the unitary fragment. Thus, using γ diagrams as a reduction class, we have shown that the unitary fragment is decidable.

Chapter 4

The decision procedure for the existential fragment

In this chapter we show that the full existential fragment is decidable by describing a decision procedure which judges the satisfiability of a generalized diagram. The procedure extends the decision procedure from the previous chapter and depends on the transformation of diagrams into a normal form. Throughout this chapter we work towards the definition of this normal form, which we call *disjunctive normal form*. The process of transforming a diagram into disjunctive normal form has several intermediate steps. Figure 4.1 illustrates these steps and the strategy used in this chapter.

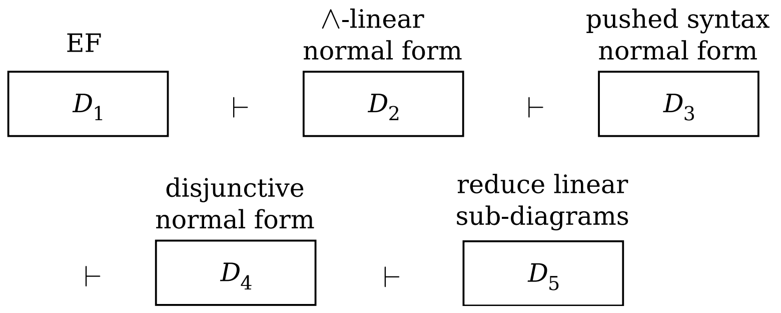


Figure 4.1: The normal forms defined in this chapter.

Diagram D_1, Figure 4.1, is a diagram from the existential fragment. Information-preserving inference rules are applied to transform D_1 into D_2, a diagram which is in $\wedge$*-linear normal form*. This means that, in D_2, all $\wedge$-labelled nodes have an out-degree of one; $\wedge$-linear normal form is defined in section 4.2. Information-preserving inference rules are then applied to transform D_2 into D_3, where D_3 is in *pushed syntax normal form.* A diagram is in pushed syntax normal form if all possible syntax is represented in its leaf nodes. This normal form is defined in section 4.3. Next, information-preserving inference rules are applied to transform D_3 into D_4, where D_4 is in disjunctive normal form, defined in section 4.4.

As we will see, a generalized diagram in disjunctive normal form is either a linear diagram or is composed of a disjunction of linear sub-diagrams. In the final step, we use inference rules to reduce each linear diagram to a generalized diagram consisting of a single node, as described in section 4.5. It is at this stage that we can make use of the decision procedure for unitary diagrams given in the previous chapter. In Figure 4.1, the meaning of D_5 is given by the disjunction of the meaning of a set of leaf nodes, and D_5 is satisfiable if and only if one or more of the unitary diagrams labelling its leaf nodes is satisfiable. The inference rules used to transform D_1 into D_5 are equivalences, and so $D_1 \equiv_{\models} D_5$. Thus, since D_1 and D_5 have all the same models, D_1 is satisfiable if and only if D_5 is satisfiable.

At each step in the process, we move from one form to the next by applying a sequence of inference rules. The way in which this is done is algorithmic; the rules are applied iteratively and the process includes choice at various stages. Thus, each section in this chapter focuses on the development of an algorithm: one which transforms generalized diagrams into $\wedge$-linear normal form, one which transforms a diagram in $\wedge$-linear normal form to a diagram in pushed syntax normal form, and so on.

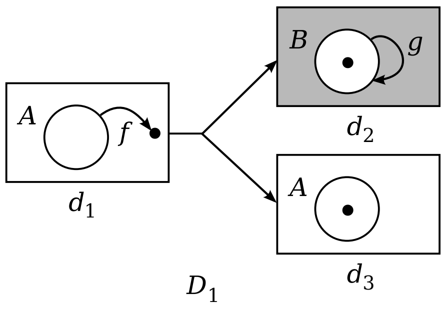


Figure 4.2: A running example: before manipulation into normal form.

To motivate the use of disjunctive normal form, consider diagram D_1 in Figure 4.2. The diagram, D_n, shown in Figure 4.3, conveys the same information as D_1 in Figure 4.2. In several cases, D_1 allows us to infer disjunctive information. For example, the unitary diagram d_2 shows a spider inhabiting a zone inside B. We can infer that this spider may or may not represent an element of A and, furthermore, that it may or may not represent the same element as the spider inside A in the unitary diagram d_3. These inferences are reified in D_n, in which the unitary diagrams each present one combination of the options. All information in D_n is held in leaf nodes, each of which forms a branch of an $\vee$-labelled connective; the only other node is labelled by a unitary diagram which is empty of syntax other than for a single zone, and is thus trivially true in any interpretation. Hence, the meaning of D_n is given by the disjunction of the meaning of its leaf nodes. It follows that we can judge the satisfiability of D_n by applying the decision procedure for unitary diagrams to its leaf nodes, and that D_n is satisfiable if and only if one or more of its leaf nodes is satisfiable.

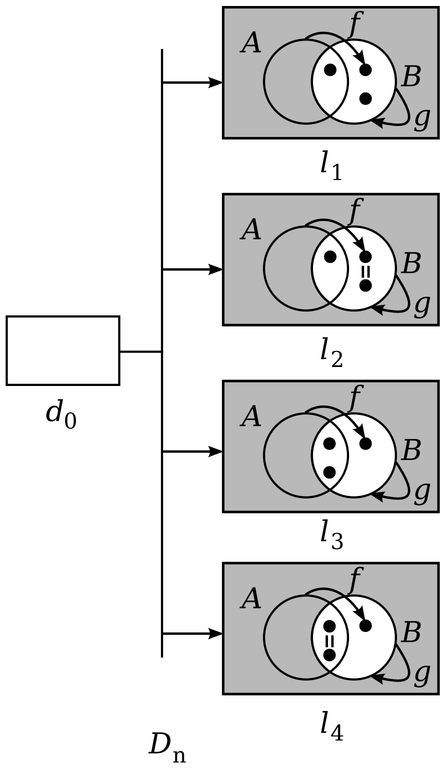


Figure 4.3: A running example: after manipulation into normal form.

We will use the diagram in Figure 4.2 as a running example throughout this chapter, showing the steps required to transform it to the diagram shown in Figure 4.3. These steps make use a series of inference rules which produce a number of intermediate, semantically equivalent normal forms, and culminate in a diagram which is in disjunctive normal form, such as diagram D_n in Figure 4.3.

We give a name to the unitary diagram which contains a single zone and is otherwise empty of syntax. Because such a diagram is trivially satisfied by any interpretation, we call it the *trivial* diagram, illustrated in Figure 4.4.

Figure 4.4: The trivial diagram.

Definition 4.0.6. Let d be the generalized unitary diagram which contains no contours, shading, spiders, ties or arrows and contains only the single zone $(\emptyset, \emptyset)$. That is,

$$d = (C, Z, Z^*, S, \eta, \tau, A),$$

where

1. $C = Z^* = S = \eta = \tau = A = \emptyset$, and
2. $Z = \{(\emptyset, \emptyset)\}$.

Then we say that d is the **trivial diagram**, denoted $\top$.

In section 4.1, we define a set of purely syntactic transformations for generalized (non-unitary) diagrams. Sections 4.2 and 4.3 describe the intermediate normal forms and the inference rules required to produce them. The inference rules will be defined using the diagram transformations. In section 4.4, we describe disjunctive normal form and its associated inference rules, and in section 4.5 we show how to transform a diagram in disjunctive normal form so that the decision procedure for unitary diagrams can be used to judge its satisfiability. Finally, in section 4.6 we show that the preceding results can be combined to describe a decision procedure for the existential fragment.

4.1 Transformations

Rather than dealing with the content of unitary diagrams, the transformations defined in this section are designed to alter the tree structure of generalized diagrams, whilst maintaining structural invariants such as the fact that all leaf nodes of a generalized diagram are labelled by unitary diagrams. In Figure 4.5 we can remove the largest sub-diagram of D_1 whose root node is labelled by the unitary diagram d_2, to obtain D_2, which is a well-formed generalized diagram. In Figure 4.6, removing just the node labelled by d_3 from D_2 results in D_3, a collection of nodes which is not a generalized diagram since it contains a leaf node which is not labelled by a unitary diagram.

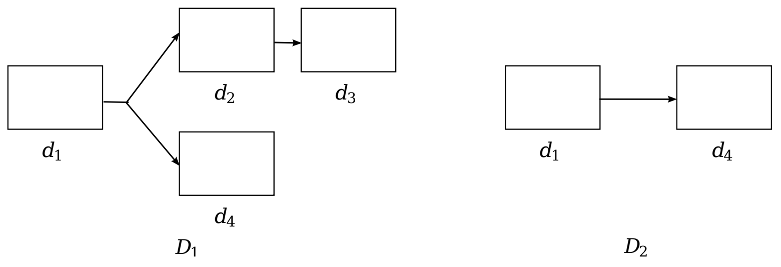


Figure 4.5: Transforming a generalized diagram by removing a sub-diagram.

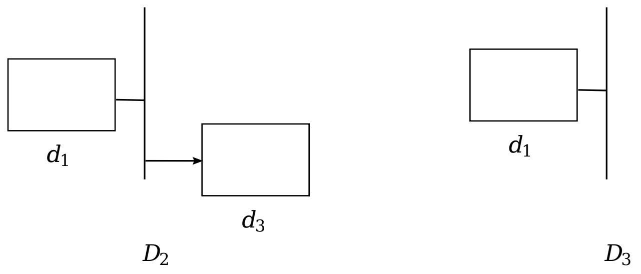


Figure 4.6: An invalid compound transformation.

In Figure 4.7, diagram D_2 is a sub-diagram of D_1. We can see that all leaf nodes of D_2 (the nodes n_7 and n_8) are also leaf nodes of D_1. In other words, D_2 is the largest sub-diagram of D_1 whose root node is n_3. We define the notion of largest sub-diagrams as the sub-diagrams *induced* by a particular node.

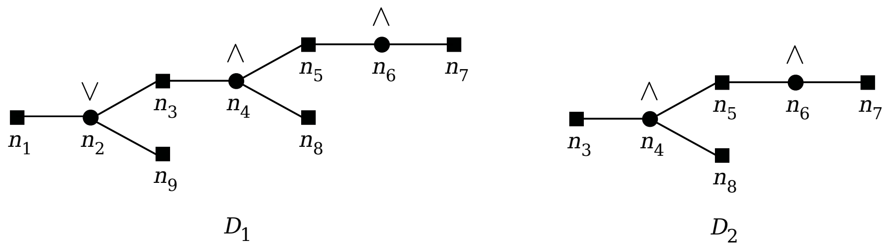


Figure 4.7: The sub-diagram induced by a particular node.

Definition 4.1.1. Let D_1 be a generalized diagram with diagram-labelled node n. We say that the sub-diagram of D_1, $D_2 = (V, W, E, l)$, with root node n and with $V \cup W = \{n\} \cup Des(n, D_1)$ is the sub-diagram of D_1 **induced** by n.

4.1.1 Atomic transformations

The first transformation we define is one which removes a sub-diagram from a generalized diagram.

Transformation 13: Remove sub-diagram. Given a generalized diagram D_1 which has a non-root, diagram-labelled node n, we can transform D_1 to obtain a second diagram, say D_2, by removing the sub-diagram induced by n. D_2 is a copy of D_1 in which n and all of its descendants have been removed and, if the immediate ancestor of n is a connective-labelled leaf node after this operation, that node is also removed. In Figure 4.8, diagram D_2 is the result of removing from diagram D_1 the sub-diagram of D_1 induced by the node n_5.

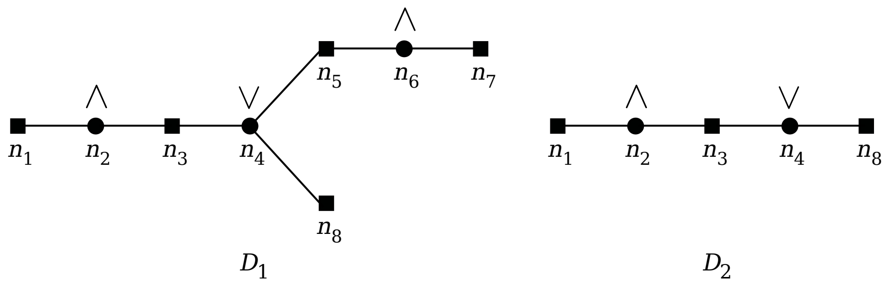


Figure 4.8: Removing a sub-diagram.

Formal description. Let $D_1 = (V_1, W_1, E_1, l_1)$ and $D_1' = (V_1', W_1', E_1', l_1')$ be generalized diagrams and let n and n_A be nodes such that the following is true:

1. n is a diagram-labelled node in D_1 and is not the root of D_1,
2. n_A is the immediate ancestor of n in D_1, and
3. D_1' is the sub-diagram of D_1 induced by n.

Let $D_2 = (V_2, W_2, E_2, l_2)$ be the generalized diagram which satisfies the following:

1. $V_2 = V_1 - V_1'$.
2. If n_A has more than one immediate descendant in D_1, then
$$W_2 = W_1 - W_1'.$$
Otherwise,
$$W_2 = W_1 - (\{n_A\} \cup W_1').$$
3. If n_A has more than one immediate descendant in D_1, then
$$E_2 = E_1 - (\{(n_A, n)\} \cup E_1').$$
Otherwise,
$$E_2 = E_1 - (\{(n', n_A) \in E_1\} \cup \{(n_A, n)\} \cup E_1').$$
4. $l_2 = l_1|_{V_2 \cup W_2}$.

Then we say we can obtain D_2 from D_1 and D_1' using the **remove sub-diagram** transformation, denoted $D_1 \xrightarrow{-D_1'} D_2$.

Next we define two transformations that allow us to remove a single diagram-labelled node, provided that the result is a generalized diagram. For the simplicity of the definitions, we separate the cases of removing root and non-root nodes.

Transformation 14: Remove root node. In Figure 4.9, we can transform D_1 by removing the root node, n_1, giving diagram D_2. Note that the immediate descendant of n_1, n_2, is also removed. It follows that this transformation is only possible when the immediate descendant of the root node is a linear connective.

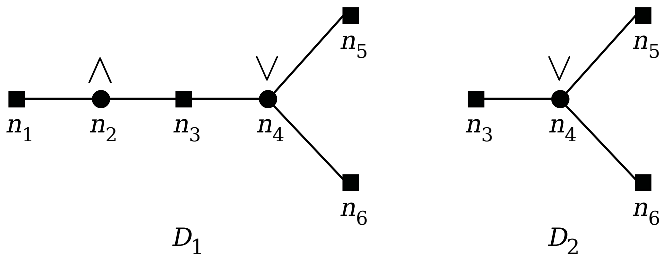


Figure 4.9: Transforming a diagram by removing the root node.

Formal description. Let $D_1 = (V, W, E, l)$ be a generalized diagram that contains nodes n_1, and n_2 which satisfy the following:

1. n_1 is the root node of D_1, and
2. n_2 is the unique immediate diagram-labelled descendant of n_1.

Let D_2 be the sub-diagram of D_1 induced by n_2. Then we say that we can obtain D_2 from D_1 and n_1 using the **remove root node** transformation, denoted $D_1 \xrightarrow{-n_1} D_2$.

Transformation 15: Remove diagram-labelled node. In Figure 4.10 we can transform D_1 by removing the node n_3, giving D_2. Note that the immediate ancestor of n_3 in D_1, n_2, is not present in D_2. The node n_2 is removed to maintain the bipartite tree of the diagram (i.e. alternating diagram- and connective-labelled nodes).

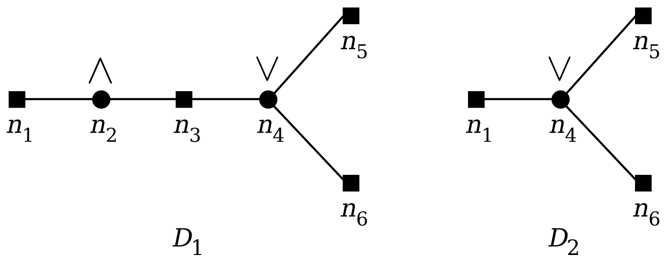


Figure 4.10: Transforming a diagram by removing a non-root node.

If the immediate ancestor of the node to be removed is non-linear, however, it is not removed as part of the transformation. In Figure 4.11, the node n_4 is removed from D_1 to give D_2. In this case it is the immediate descendant of n_4,

n_5, which is removed to maintain the bipartite tree. It follows that, to remove a diagram-labelled node n, either one or both of the immediate relatives of n must be linear. If both immediate relatives of n are linear, the transformation removes n and its immediate ancestor; an equivalent diagram would be produced if the transformation were defined to remove n and its immediate descendant in this case.

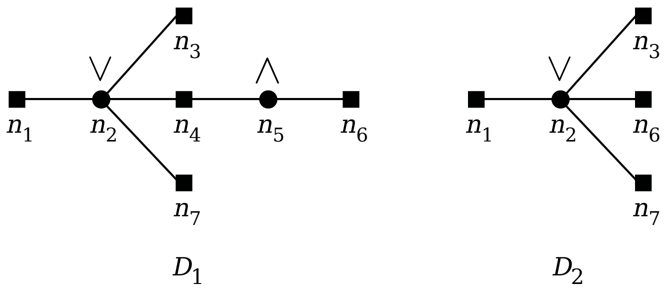


Figure 4.11: One of the immediate neighbours must be linear when removing a node.

Formal description. Let $D_1 = (V_1, W_1, E_1, l_1)$ be a generalized diagram that contains nodes n_1, n_2 and n_3 which satisfy the following:

1. n_2 is a diagram-labelled node in D_1 which is not the root node of D_1,
2. n_1 is the immediate ancestor of n_2,
3. n_3 is the immediate descendant of n_2, and
4. either n_1 is a linear connective, or n_3 is a linear connective, or both.

Let n_0 be the immediate diagram-labelled ancestor of n_2 and let $D_2 = (V_2, W_2, E_2, l_2)$ be the diagram which satisfies the following:

1. $V_2 = V_1 - \{n_2\}$.
2. If n_1 is a linear connective, $W_2 = W_1 - \{n_1\}$. Otherwise, n_3 is a linear connective and we define $W_2 = W_1 - \{n_3\}$.
3. If n_1 is a linear connective, then E_2 is defined as follows:

$$E_2 = \big(E_1 - \{(n_0, n_1), (n_1, n_2), (n_2, n_3)\}\big) \cup \{(n_0, n_3)\}.$$

Otherwise, n_3 is a linear connective: let n_4 be the (unique) immediate descendant of n_3. Then E_2 is defined as follows:

$$E_2 = (E_1 - \{(n_1, n_2), (n_2, n_3)\}) \cup \{(n_1, n_4)\}.$$

4. $l_2 = l_1|_{V_2 \cup W_2}$.

Then we say that we can obtain D_2 from D_1 and n_2 using the **remove diagram-labelled node** transformation, denoted $D_1 \xrightarrow{-n_2} D_2$.

Next, we define two transformations which attach diagrams. The first attaches a diagram to a leaf node and requires that a new connective be supplied, and the second attaches a diagram to an existing connective-labelled node.

In the following transformation, and in several more below, we use the $\uplus$ symbol, denoting the disjoint union of sets. When we combine sets of nodes from generalized diagrams, we take the disjoint union of the sets of nodes to ensure that any necessary renaming takes place, and that they are indeed treated as disjoint sets. We assume that the labels of nodes are not affected when nodes are renamed. That is, suppose we have a transformation T that somehow combines generalized diagrams $D_1 = (V_1, W_1, E_1, l_1)$ and $D_2 = (V_2, W_2, E_2, l_2)$ to produce a third diagram $D_3 = (V_3, W_3, E_3, l_3)$ where $V_3 \subseteq V_1 \uplus V_2$ and $W_3 \subseteq W_1 \uplus W_2$. Furthermore, suppose there is a node, n, in D_1 which is renamed n' in D_3 by the disjoint union operation. Then, unless T explicitly includes a relabelling operation, $l_1(n) = l_3(n')$.

Transformation 16: Attach diagram to leaf. Figure 4.12 shows three diagrams, D_1, D_2 and D_3, where D_3 is the result of attaching D_2 to the leaf node of D_1, n_3. Note that D_3 contains a node, labelled m, which is not in D_1 or D_2.
Formal description. Let $D_1 = (V_1, W_1, E_1, l_1)$ and $D_2 = (V_2, W_2, E_2, l_2)$ be generalized diagrams where D_1 has a leaf node n and let $\diamond$ be a connective where $\diamond \in \{\wedge, \vee\}$. Choose a node, m, not in D_1 or D_2 and let $D_3 = (V_3, W_3, E_3, l_3)$ be the diagram which satisfies the following:

1. $V_3 = V_1 \uplus V_2$,

2. $W_3 = W_1 \uplus \{m\} \uplus W_2$,

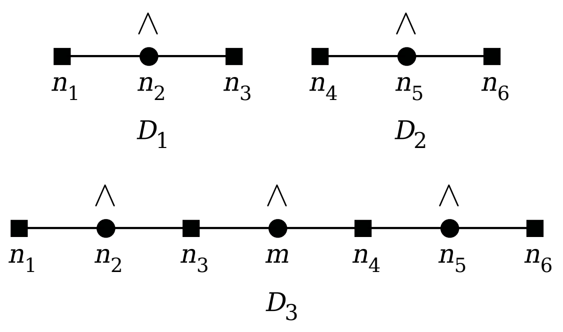


Figure 4.12: Attaching a diagram to a leaf node.

3. $E_3 = E_1 \uplus \{(n, m), (m, root(D_2))\} \uplus E_2$,

4. $l_3 = l_1 \uplus \{(m, \diamond)\} \uplus l_2$.

Then we say we can obtain D_3 from D_1, n, $\diamond$ and D_2 using the **attach diagram to leaf** transformation, denoted $D_1 \xrightarrow{+(n,\diamond,D_2)} D_3$.

Transformation 17: Attach diagram to connective. As well as attaching diagrams to leaf nodes, we can attach diagrams to an existing connective-labelled node. In this case there is no need to specify a logical connective and the resulting diagram contains only those nodes which are in the initial diagrams. Figure 4.13 shows three diagrams, D_1, D_2 and D_3, where D_3 is the result of attaching D_2 to the connective-labelled node n_2 in D_1.

Formal description. Let $D_1 = (V_1, W_1, E_1, l_1)$ and $D_2 = (V_2, W_2, E_2, l_2)$ be generalized diagrams where D_1 has a connective-labelled node n. Let $D_3 = (V_3, W_3, E_3, l_3)$ be the diagram which satisfies the following:

1. $V_3 = V_1 \uplus V_2$,

2. $W_3 = W_1 \uplus W_2$,

3. $E_3 = E_1 \uplus \{(n, root(D_2))\} \uplus E_2$,

4. $l_3 = l_1 \uplus l_2$.

Then we say we can obtain D_3 from D_1, n and D_2 using the **attach diagram to connective** transformation, denoted $D_1 \xrightarrow{+(n,D_2)} D_3$.

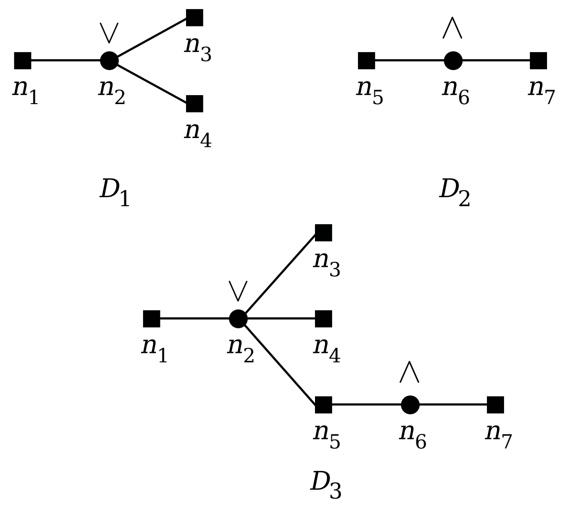


Figure 4.13: Attaching a diagram to a connective-labelled node.

Transformation 18: Relabel node. Given a generalized diagram D, we can relabel a node in D by making a change to its labelling function. No changes are required to the nodes or edges of D. Figure 4.14 shows two diagrams, D_1 and D_2, in which the node n_3 in D_1 is labelled by the unitary diagram d_1. Diagram D_2 shows the result of relabelling n_3 in D_1 by the unitary diagram d_2. We can also relabel connective-labelled nodes.

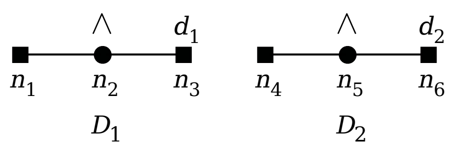


Figure 4.14: Relabelling a node.

Formal description. Let $D_1 = (V_1, W_1, E_1, l_1)$ be a generalized diagram which includes a node n and, if n is a diagram-labelled node, let λ be a unitary diagram. Otherwise, n is a connective-labelled node; let λ be one of $\wedge$ or $\vee$. Let $D_2 = (V_1, W_1, E_1, l_2)$ be the diagram where

$$l_2 = (l_1 - \{(n, l_1(n))\}) \cup \{(n, \lambda)\}.$$

Then we say we can obtain D_2 from D_1, n and λ using the **relabel node** transformation, denoted $D_1 \xrightarrow{l(n,\lambda)} D_2$.

To conclude this section, we summarise the atomic transformations in the following table:

Table 4.1: The atomic compound transformations.

Transformation	**Notation**	**Parameters**	**Description**
13: Remove sub-diagram.	$D_1 \xrightarrow{-D'_1} D_2$	D'_1 is the sub-diagram of D_1 induced by a particular node.	Removes D'_1 from D_1 to give D_2.
14: Remove root node.	$D_1 \xrightarrow{-n_1} D_2$	n_1 is the root node of D_1.	Removes n_1 from D_1 to give D_2.
15: Remove diagram-labelled node.	$D_1 \xrightarrow{-n} D_2$	The node n is a diagram-labelled node of D_1.	Removes n from D_1 to give D_2.
16: Attach diagram to leaf.	$D_1 \xrightarrow{+(n,\diamond,D_2)} D_3$	The node n is a leaf node of D_1, $\diamond$ is one of the logical connectives $\wedge$ or $\vee$, and D_2 is a generalized diagram.	Attaches D_2 to n in D_1 using $\diamond$, giving D_3.
17: Attach diagram to connective.	$D_1 \xrightarrow{+(n,D_2)} D_3$	The node n is a connective-labelled node in D_1, and D_2 is a generalized diagram.	Attaches D_2 to n in D_1, giving D_3.

Continued on next page . . .

Table 4.1 – Continued

Transformation	Notation	Parameters	Description
18: Relabel node.	$D_1 \xrightarrow{l(n,\lambda)} D_3$	The node n is a node in D_1 and, if n is a connective-labelled node then λ is one of the logical connectives $\wedge$ or $\vee$ and, if n is a diagram-labelled node, then λ is a unitary diagram.	Relabels n by λ in D_1, giving D_3.

4.1.2 Derived transformations

We can now define a series of derived transformations which make use of the atomic transformations from the previous section. The first two such transformations attach a set of diagrams to connective-labelled nodes and leaf nodes respectively.

Transformation 19: Attach diagrams to connective. In Figure 4.15, diagram D_4 shows the result of attaching D_2 and D_3 to the connective-labelled node n_4 in D_1.

Formal description. Let D_1 be a generalized diagram which has a connective-labelled node n and let $\mathcal{D} = \{A_1, \ldots, A_m\}$ be a non-empty set of generalized diagrams. Let D_2 be the diagram obtained by repeated use of transformation 17, attach diagram to connective, as follows:

$$D_1 \xrightarrow{+(n,A_1)} D_{A_1} \ldots D_{A_{m-1}} \xrightarrow{+(n,A_m)} D_2.$$

Then we say we can obtain D_2 from D_1, n and $\mathcal{D}$ using the **attach diagrams to connective** transformation, denoted $D_1 \xrightarrow{+(n,\mathcal{D})} D_2$.

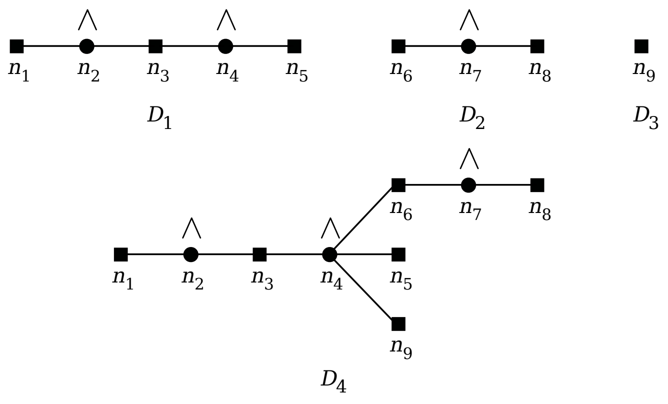


Figure 4.15: Attaching a set of diagrams to a connective-labelled node.

Transformation 20: Attach diagrams to leaf. When we attach a series of diagrams, $\mathcal{D} = \{A_1, \dots, A_n\}$ to the leaf node, n, of a diagram, D_1, the first operation is to attach A_1 to n using transformation 16, attach diagram to leaf, giving a diagram, say D_1'. The node n is not a leaf node in D_1', and so the remaining transformations, those which attach the diagrams in the set $\mathcal{D} - \{A_1\}$ to n, use transformation 19, attach diagrams to connective. Figure 4.16 shows the steps involved in attaching diagrams D_2 and D_3 to the leaf node, n_3, of D_1 using the $\vee$ connective. Diagram D_1' shows the result of attaching D_2 to n_3 in D_1 using $\vee$ and transformation 16, attach diagram to leaf. This results in a new node, m. Diagram D_4 shows the result of attaching D_3 to m in D_1' using transformation 19, attach diagrams to connective.

Formal description. Let D_1 be a generalized diagram which has a leaf node n, let $\mathcal{D} = \{A_1, \dots, A_m\}$ be a non-empty set of generalized diagrams and let $\diamond$ be a connective where $\diamond \in \{\wedge, \vee\}$. Let D_{A_1} be the diagram obtained by attaching A_1 to n using $\diamond$ and transformation 16, attach diagram to leaf: $D_1 \xrightarrow{+(n,\diamond,A_1)} D_{A_1}$. Let n_D be the immediate descendant of n in D_{A_1} and let D_2 be the diagram obtained by attaching $\mathcal{D} - \{A_1\}$ to n_D in D_{A_1} using transformation 19, attach diagrams to connective: $D_{A_1} \xrightarrow{+(n_D,\mathcal{D}-\{A_1\})} D_2$. Then we say we can obtain D_2 from D_1, n, $\diamond$ and $\mathcal{D}$ using the **attach diagrams to leaf** transformation, denoted $D_1 \xrightarrow{+(n,\diamond,\mathcal{D})} D_2$.

Transformation 21: Attach diagrams to leaves. We will sometimes need to attach a diagram or set of diagrams to every leaf node in a diagram. Figure 4.17

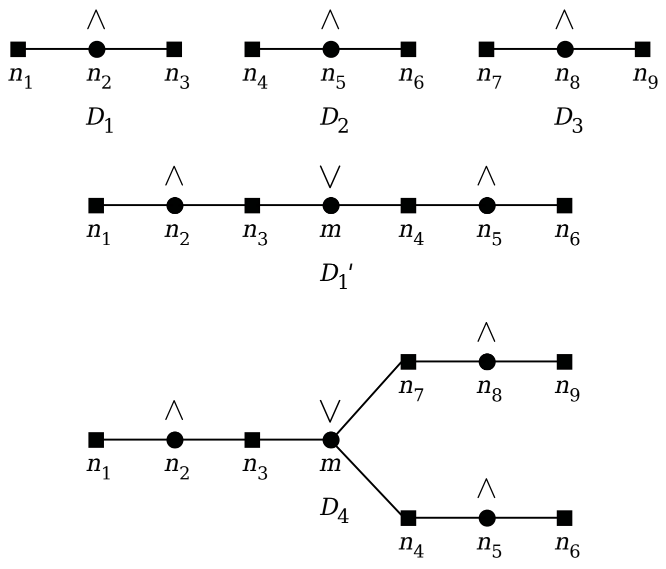


Figure 4.16: Attaching a set of diagrams to a leaf node.

shows three diagrams, D_1, D_2 and D_3. The result of attaching the set of diagrams $\{D_2, D_3\}$ to every leaf node in D_1 using the $\wedge$ connective is shown in Figure 4.18.

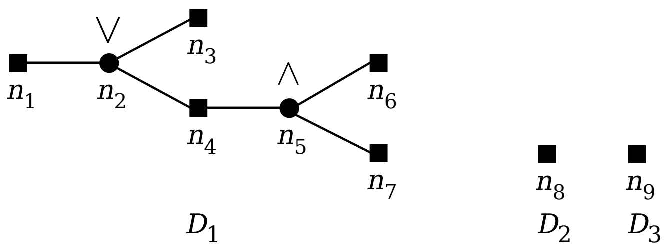


Figure 4.17: Prior to attaching a set of diagrams to every leaf node.

In Figure 4.18, diagram D_4, the leaf nodes are copies of diagrams D_2 and D_3, Figure 4.17. Where a leaf node is labelled, for example, n_8^+, the + indicates that this node is a copy of n_8 which has been renamed as necessary to make it unique in D_4.

Formal description. Let D_1 be a generalized diagram, let $\mathcal{D}$ be a non-empty set of generalized diagrams and let $\diamond \in \{\wedge, \vee\}$. Let D_2 be a copy of D_1 in which $\mathcal{D}$ is attached to each leaf node of D_1 using $\diamond$ and transformation 20, attach diagrams to leaf. That is, if D_1 has leaf nodes $\mathcal{L} = \{l_1, \ldots, l_m\}$ then D_2 is produced as

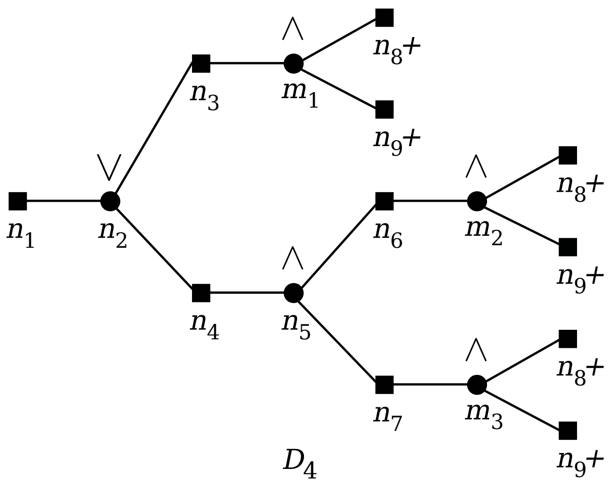


Figure 4.18: The result of attaching a set of diagrams to every leaf node.

follows:

$$D_1 \xrightarrow{+(l_1,\diamond,\mathcal{D})} D_{l_1} \ldots D_{l_{m-1}} \xrightarrow{+(l_m,\diamond,\mathcal{D})} D_2.$$

Then we say we can obtain D_2 from D_1 and D using the **attach diagrams to leaves** transformation, denoted $D_1 \xrightarrow{+(\diamond,\mathcal{D})} D_2$.

Transformation 22: Replace sub-diagram. Next, we define a transformation that, given a diagram D_1, replaces a sub-diagram of D_1 by another diagram, D_2 In Figure 4.19, diagram D_3 shows the result of replacing the sub-diagram of D_1 induced by node n_3, by diagram D_2. In effect, this composes the transformations remove sub-diagram and attach sub-diagram.

Formal description. Let D_1, D_2 and D_3 be generalized diagrams such that D_2 is the sub-diagram of D_1 induced by a diagram-labelled node, n_3. Let D_1' be the diagram obtained by removing D_2 from D_1 using transformation 13, remove sub-diagram: $D_1 \xrightarrow{-D_2} D_1'$.

1. If the immediate ancestor of n_3, n_2, is present in D_1', let D_4 be the diagram obtained by using transformation 17, attach diagram to connective, to attach D_3 to n_2 in D_1': $D_1' \xrightarrow{+(n_2,D_3)} D_4$.

2. Otherwise, the immediate ancestor of n_3 is not present in D_1'. Let n_1 be the immediate diagram-labelled ancestor of n_3 in D_1 and let D_4 be the diagram obtained by using transformation 16, attach diagram to leaf, to attach D_3

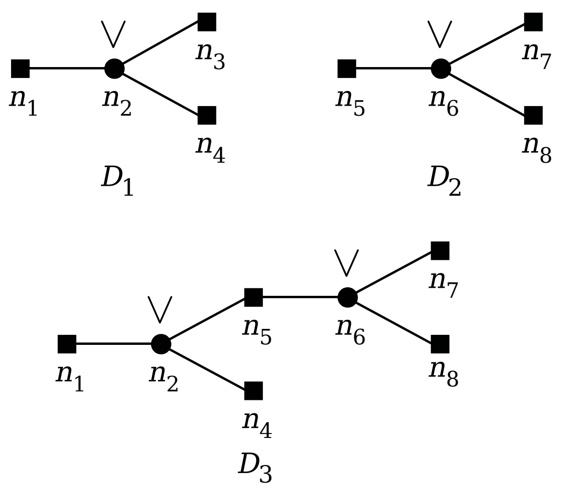


Figure 4.19: Replacing a sub-diagram.

to n_1 in D_1' using $\wedge$: $D_1' \xrightarrow{+(n_1,\wedge,D_3)} D_4$.

Then we say we can obtain D_4 from D_1, D_2 and D_3 using the **replace sub-diagram** transformation, denoted $D_1 \xrightarrow{\rho(D_2,D_3)} D_4$.

Finally, we define a transformation which allows us to replace an inner (i.e. non-leaf) node in a diagram by another diagram. Figure 4.20 depicts three diagrams: D_1, D_2, and D_3. Figure 4.21 shows the result of transforming D_1,

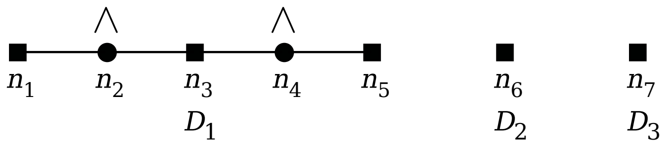


Figure 4.20: Preparing to replace a sub-diagram by a set of diagrams.

Figure 4.20, by replacing the node n_3 with the disjunction of the diagrams D_2 and D_3 from Figure 4.20. Note that the diagram-labelled ancestors of n_3 in D_1, Figure 4.20, which we can think of as the prefix of the node to be replaced, are maintained in D_4, Figure 4.21. Also, the diagram-labelled descendants of n_3, which we can think of as the suffix of the node to be replaced, are attached to each leaf node in the replacement. If we think of diagram D_1, Figure 4.20, as composed of the node n_3 with its prefix and suffix, the prefix and suffix are maintained in diagram D_4, Figure 4.21, whilst n_3 itself is replaced by the disjunction of diagrams D_2 and D_3, Figure 4.20. In diagram D_4, Figure 4.21, the nodes which are created during the transformation are labelled m_1, m_2 and m_3, and the two nodes labelled n_5^+ are copies of node n_5 from diagram D_1, Figure 4.20.

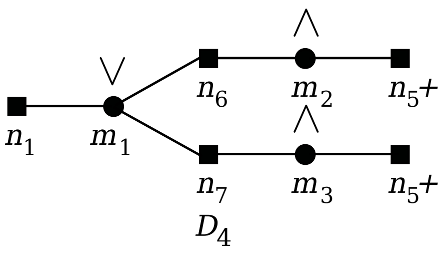


Figure 4.21: The result of replacing a sub-diagram.

We present a second, more complex, example of replacing an inner node, which illustrates the case when the immediate ancestor of the node to be replaced is a non-linear connective. Figure 4.22 shows an initial diagram, D_1, in which the node to be replaced, n_3, is highlighted, and a set of diagrams, $\mathcal{D}_1$, with which n_3 will be replaced.

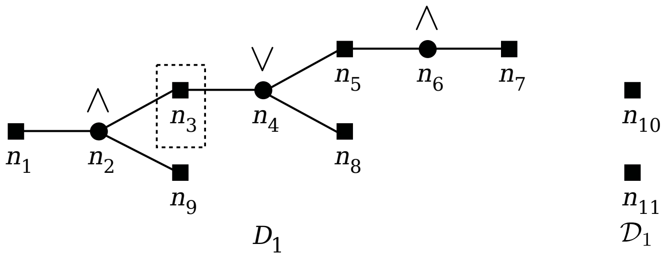


Figure 4.22: Replacing an inner node.

In Figure 4.23, diagram D' is the sub-diagram of D_1, Figure 4.22, induced by n_3. Diagram D'_1 is the diagram given by removing D' from D_1. We can think of D'_1 as the prefix of n_3.

In Figure 4.24, the set of diagrams $\mathcal{D}_2$ is the set of sub-diagrams of D_1, Figure 4.22, induced by the immediate descendants of n_3. These diagrams are the suffix of the node to be replaced, n_3. Next, we attach the suffix of n_3 to the diagrams which will be used to replace n_3. That is, the set of diagrams $\mathcal{D}_3$ is formed by attaching the diagrams in $\mathcal{D}_2$ to the diagrams in the set $\mathcal{D}_1$. In doing this, we use the connective labelling the immediate descendant of the node to

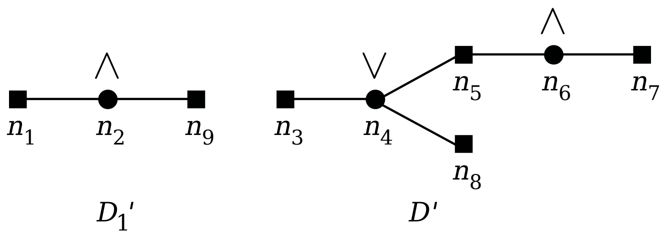


Figure 4.23: Replacing an inner node: the first intermediate steps.

be replaced, node n_4 in diagram D_1, Figure 4.22. Node n_4 is labelled by $\vee$, and so copies of the diagrams in $\mathcal{D}_2$, Figure 4.24, are attached to the leaves of the diagrams in $\mathcal{D}_1$, Figure 4.22, using $\vee$, to form the set of diagrams $\mathcal{D}_3$.

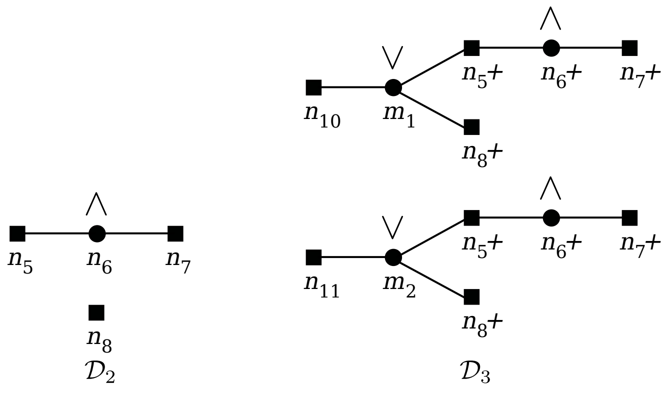


Figure 4.24: Replacing an inner node: the next intermediate steps.

Finally, we attach the set of diagrams, $\mathcal{D}_3$, to n_2 in diagram D_1', Figure 4.23, which is the immediate ancestor of n_3, the node to be replaced. The result of this operation is shown in Figure 4.25, diagram D_2.

Transformation 23: Replace inner node. Since we will only use the following transformation to replace non-leaf nodes, we define it in this context to keep the definition simpler.

Formal description. Let D_1 be a generalized diagram with nodes n_1, n_2 and n_3 such that:

1. n_1 and n_3 are diagram-labelled nodes,

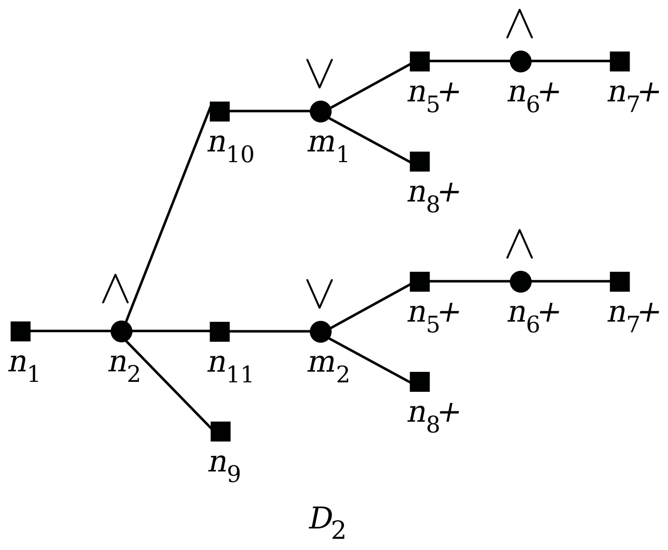


Figure 4.25: The result of replacing an inner node.

2. n_2 is a connective-labelled node and is the immediate descendant of n_1 and the immediate ancestor of n_3,

3. n_3 is not a leaf node.

Let $\mathcal{D}_1$ be a non-empty set of generalized diagrams and let $\diamond \in \{\wedge, \vee\}$. Let D' be the sub-diagram of D_1 induced by n_3 and let D'_1 be the diagram obtained by using transformation 13, remove sub-diagram, to remove D' from D_1: $D_1 \xrightarrow{-D'} D'_1$. Let $\diamond'$ be the connective which labels the immediate descendant of n_3 in D_1. Let $\mathcal{D}_2$ be the set of sub-diagrams of D_1 induced by the immediate diagram-labelled descendants of n_3. Let $\mathcal{D}_3$ be the set of diagrams obtained by attaching $\mathcal{D}_2$ to the leaf nodes of each diagram in $\mathcal{D}_1$ using $\diamond'$ and transformation 21, attach diagrams to leaves:

$$\mathcal{D}_3 = \{D_i \xrightarrow{+(\diamond', \mathcal{D}_2)} D'_i : D_i \in \mathcal{D}_1\}.$$

1. If the node n_2 is present in D'_1, then let D_2 be the diagram obtained by attaching $\mathcal{D}_3$ to n_2 in D'_1 using transformation 19, attach diagrams to connective: $D'_1 \xrightarrow{+(n_2, \mathcal{D}_2)} D_2$.

2. Otherwise, the node n_2 is not present in D'_1. Let D_2 be the diagram obtained by attaching $\mathcal{D}_2$ to n_1 in D'_1 using $\diamond$ and transformation 20, attach diagrams to leaf: $D'_1 \xrightarrow{+(n_1, \diamond, \mathcal{D}_2)} D_2$.

Then we say we can obtain D_2 from D_1, n_3, $\diamond$ and $\mathcal{D}_1$ using the **replace inner node** transformation, denoted $D_1 \xrightarrow{\rho(n_3,\diamond,\mathcal{D}_1)} D_2$.

This completes the set of compound transformations. As we have seen, the majority of the transformations are derived from simpler, atomic operations. In the previous section we showed that we have a complete set of unitary transformations, sufficient to transform a unitary diagram to any other. Although we do not give a corresponding proof for the compound transformations, we will sketch the way in which this could be done. To be syntactically complete, the compound transformations must be sufficient to transform an initial diagram, D_1, into a target diagram D_2. In Figure 4.26, suppose we wish to transform D_1 into D_2. We begin by using transformation 13, remove sub-diagram, to remove everything except the root node of D_1, giving D_{11}. To this diagram, we attach a copy of D_2, using transformation 16, attach diagram to leaf, and use transformation 18, relabel node, to relabel n_1 with the unitary diagram labelling the root of D_2, giving diagram D_{12}. Finally, we use transformation 15, remove node, to remove from D_{12} the node which was the root node of D_2, giving diagram D_{13}.

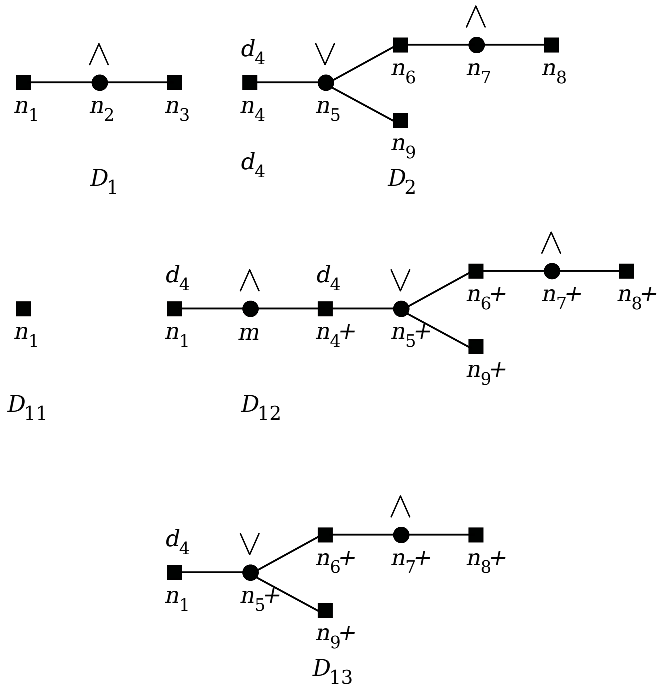


Figure 4.26: The completeness of the compound transformations.

As with the atomic transformations, we conclude with a table of the derived transformations:

Table 4.2: The derived compound transformations.

Transformation	Notation	Parameters	Description
19: Attach diagrams to connective.	$D_1 \xrightarrow{+(n,\mathcal{D})} D_2$	The node n is a connective-labelled node in D_1, and $\mathcal{D}$ is a non-empty set of generalized diagrams.	Attaches the diagrams in $\mathcal{D}$ to n in D_1, giving D_2.
20: Attach diagrams to leaf.	$D_1 \xrightarrow{+(n,\mathcal{D})} D_2$	The node n is a leaf node in D_1, and $\mathcal{D}$ is a non-empty set of generalized diagrams.	Attaches the diagrams in $\mathcal{D}$ to n in D_1, giving D_2.
21: Attach diagrams to leaves.	$D_1 \xrightarrow{+(\diamond,\mathcal{D})} D_2$	The logical connective $\diamond$ is either $\wedge$ or $\vee$, and $\mathcal{D}$ is a non-empty set of generalized diagrams.	Attaches the diagrams in $\mathcal{D}$ to every leaf node in D_1 using $\diamond$ as the connective, giving D_2.
22: Replace sub-diagram.	$D_1 \xrightarrow{\rho(D_1',D_2)} D_3$	D_1' is the sub-diagram of D_1 induced by a node, n, and D_2 is a generalized diagram.	Removes D_1' from D_1 and replaces it with D_2, giving D_3.
23: Replace inner node.	$D_1 \xrightarrow{\rho(n,\diamond,\mathcal{D})} D_2$	The node n is a diagram-labelled node in D_1, the logical connective $\diamond$ is either $\wedge$ or $\vee$, and $\mathcal{D}$ is a non-empty set of generalized diagrams.	Removes n from D_1 and uses $\diamond$ to replace it by diagrams in $\mathcal{D}$, giving D_3.

4.2 ∧-linear normal form

In this section we show that we can reduce the out-degree of a non-linear ∧-labelled connective, n, from a generalized diagram, D, without changing its meaning. This is achieved by an inference rule which reduces the out-degree of n by one. Our objective is to 'linearise' all ∧-labelled nodes in a diagram, i.e. to ensure that every ∧-labelled node has an out-degree of one. We say that diagrams in this form are in ∧-*linear normal form*.

Definition 4.2.1. Let D be a generalized diagram. If D contains no non-linear ∧-labelled connectives, we say that D is in **∧-linear normal form**.

Figure 4.27 places this section in the larger context of the decision procedure, where the dashed box labelled algorithm 1 indicates the current stage.

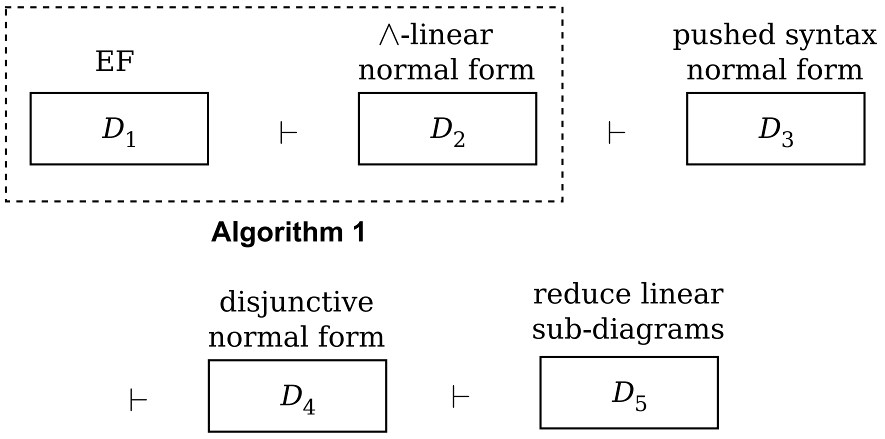


Figure 4.27: ∧-linear normal form in the context of the decision procedure.

Figure 4.28 returns to the running example introduced in Figure 4.2, page 103. Unlike the diagram in Figure 4.2, diagram D_1 in Figure 4.28 is a meta-diagram annotated with a label for the connective node, n, and labels for the spiders, 1, 2 and 3.

The formula for D_1 is given by $form(d_1) \wedge (form(d_2) \wedge form(d_3))$. The brackets in this formula are redundant, and so it is equivalent to the formula for the

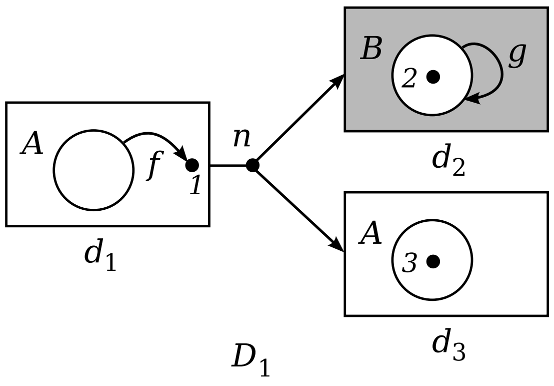


Figure 4.28: A running example: prior to linearising an $\wedge$-labelled connective.

diagram in Figure 4.29, given by $form(d_1) \wedge form(d_2) \wedge form(d_3)$. Thus, we can transform D_1, Figure 4.28, into D_2, Figure 4.29, in which all $\wedge$-labelled nodes are linear, without changing the meaning of the initial diagram. Note that this transformation results in a new node in D_2, which is labelled m.

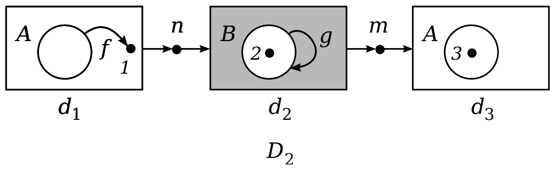
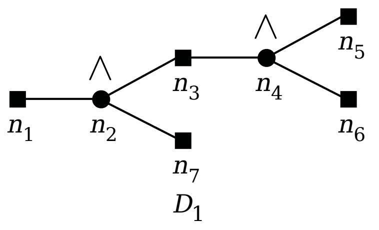


Figure 4.29: A running example: after linearising an $\wedge$-labelled connective.

When there is more than one non-linear $\wedge$-labelled connective to be linearised, the order in which we do this may have a bearing on the structure, but not the meaning, of the final result. In Figure 4.30, diagram D_1 contains two non-linear $\wedge$-labelled nodes, labelled n_2 and n_4. If we choose to linearise n_2 first, the result is the diagram D_2, Figure 4.30. Note that we have removed the lower branch of n_2 from diagram D_1 and attached a copy of it to each leaf node of the upper branch, resulting in the addition of two new connective-labelled nodes, m_1 and m_2. Equally, we could have reversed this by removing the upper branch of n_2 and attaching it to n_7, the single leaf node of the lower branch of n_2, without changing the meaning of the original diagram.

If, instead, we begin by removing one of the branches of n_4 from D_1, Figure 4.30, the result is diagram D_3, Figure 4.31. Note that, in this case, only



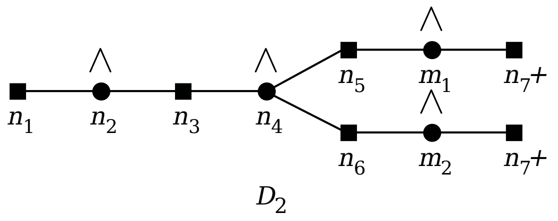


Figure 4.30: Linearising ∧-labelled connectives.

one new connective-labelled node, labelled m_1, is created. Taking the order of

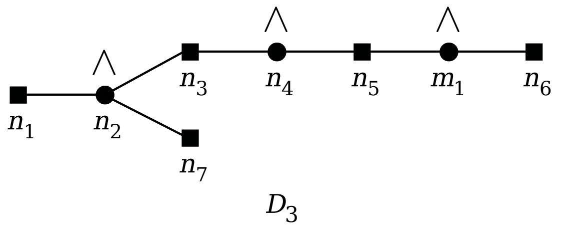


Figure 4.31: An alternative order in which to linearise ∧-labelled connectives.

operations which produces diagram D_3, Figure 4.31, and continuing to flatten non-linear ∧-labelled connectives until there are none left, produces diagram D_4, Figure 4.32. If instead, we take the order of operations that produces diagram D_2, Figure 4.30, and continue, the result is diagram D_5, Figure 4.32. Both D_4 and D_5 are linear diagrams whose meanings are given by the conjunction of the meanings of the same set of unitary diagrams. For this reason, we can see, intuitively, that their meanings are equivalent. Note, however, that diagram D_5 contains two copies of the diagram-labelled node n_7.

Rule 4: Reduce out-degree of conjunct. The following rule reduces the number of sub-diagrams attached to a non-linear ∧-labelled connective by one. To fully linearise an ∧-labelled node which has more than two branches, i.e. to reduce the out-degree to one, we apply the rule repeatedly.

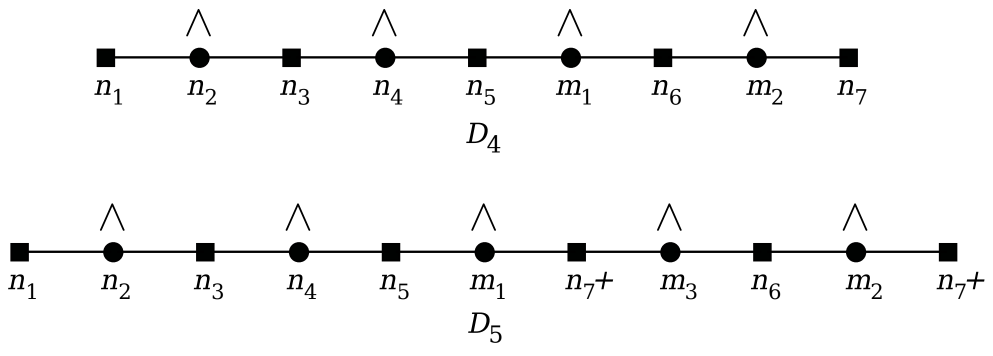


Figure 4.32: The result of linearising all $\wedge$-labelled connectives in a diagram.

Formal description. Let D_1 be a generalized diagram which includes a non-linear $\wedge$-labelled node, n. Let $\mathcal{N}$ be the set of immediate descendants of n in D_1, let $n_x \in \mathcal{N}$, and let X be the sub-diagram of D_1 induced by n_x. Let $\mathcal{Y}$ be the set of sub-diagrams of D_1 induced by the nodes in the set $\mathcal{N} - \{n_x\}$ and let $\mathcal{L}$ be the set of leaf nodes of the diagrams in $\mathcal{Y}$. Let D_1' be the diagram obtained by removing X from D_1: $D_1 \xrightarrow{-X} D_1'$. Let D_2 be the diagram obtained by attaching X to each node in $\mathcal{L}$ in D_1' using $\wedge$ and transformation 20, attach diagram to leaf:

$$D_1' \xrightarrow{+(l_1,\wedge,X)} D_{l_1} \dots D_{l_{m-1}} \xrightarrow{+(l_m,\wedge,X)} D_2,$$

for l_1 to l_m in $\mathcal{L}$. Then D_1 can be replaced by D_2 and vice versa.

Theorem 4.2.1: Rule 4, reduce out-degree of conjunct, is valid. Let D_1 be a generalized diagram which includes a non-linear $\wedge$-labelled node, n, and let D_2 be the diagram obtained by using the rule to reduce the out-degree of n by one. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be a model for D_1 with valid extension $I' = (U, \Psi', \Phi)$. As in the definition of the rule, let $\mathcal{N}$ be the set of immediate descendants of n in D_1, let $n_x \in \mathcal{N}$, and let X be the sub-diagram of D_1 induced by n_x. Let $\mathcal{Y}$ be the set of sub-diagrams of D_1 induced by the nodes in the set $\mathcal{N} - \{n_x\}$ and let $\mathcal{L}$ be the set of leaf nodes of the diagrams in $\mathcal{Y}$. Let n_0 be the immediate ancestor of n and let D_0 be the sub-diagram of D_1 induced by n_0. See Figure 4.33 for a concrete example.

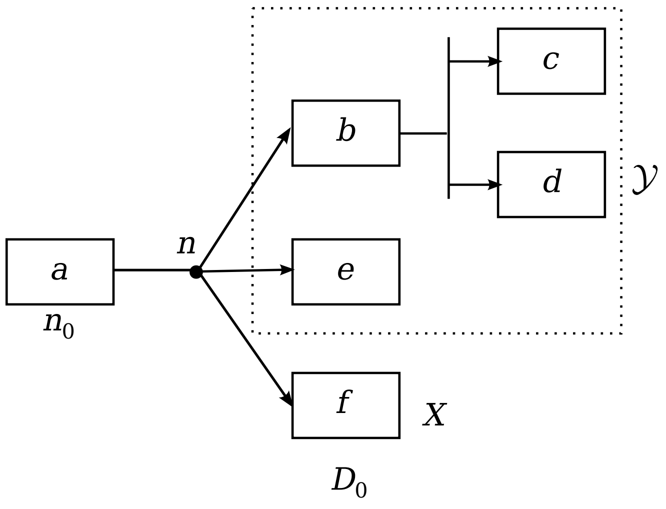


Figure 4.33: A concrete example: prior to reducing the out-degree of n.

Then the formula for D_1 contains the sub-formula for D_0, which is as follows:

$$form(d_0) \wedge \big(\bigwedge_{D' \in \{X\} \cup \mathcal{Y}} form(D') \big), \tag{4.1}$$

where d_0 is the unitary diagram labelling n_0 in D_1. In order for (4.1) to be true under I', $form(D')$ must be true for each $D' \in \{X\} \cup \mathcal{Y}$. Assume that (4.1) is true and let $D' \in \{X\} \cup \mathcal{Y}$. In the next step the sub-diagram, X, of D_1 will be removed from D_0, giving D_0', and attached to each leaf node of D_0'. Let D_0' be the diagram obtained by using transformation 13, remove sub-diagram, to remove X from D_0. Let D_0'' be the diagram obtained by using transformation 21, attach diagrams to leaves, to attach X to the leaf nodes of D_0'. Figure 4.34 shows a concrete example of this operation, in which the sub-diagram, X, of diagram D_0 in Figure 4.33 is removed and attached to the leaf nodes of the remaining diagram.

We know that $form(X)$ is true under I' since $form(D_0)$ is true and, by the same fact, that $form(Y_i)$ is true for each $Y_i \in \mathcal{Y}$. Thus, $form(D_0'')$ is true under I'. Diagram D_2 is the diagram obtained by replacing D_0 by D_0'' in D_1, and it follows that the formula for D_2 is true whenever the formula for D_1 is true. Thus, I' is valid for D_2 and $D_1 \vDash D_2$. We can show that $D_2 \vDash D_1$ by a similar argument, and so $D_1 \equiv_{\vDash} D_2$ as required.

$\square$

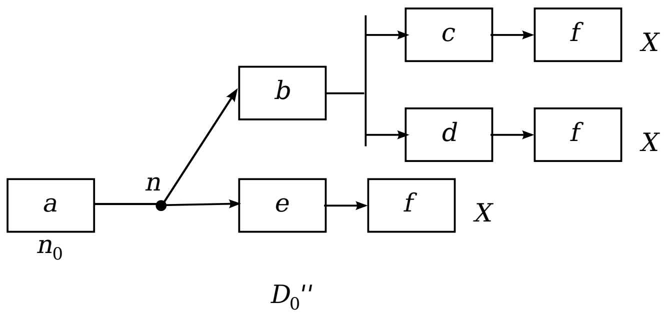


Figure 4.34: A concrete example: after reducing the out-degree of n by one.

We now state the steps required to transform a generalized diagram to ∧-linear normal form, which is done using a finite sequence of applications of rule 4, reduce out-degree of conjunct.

Algorithm 1 (Generalized diagram to ∧-linear normal form)**.** Let D_1 be a generalized diagram which contains a non-linear ∧-labelled node, n. Use rule 4, reduce out-degree of conjunct, to reduce the out-degree of n by one, giving diagram D_1'. Repeat this step until we obtain a diagram, D_2, in ∧-linear normal form.

Before showing that the above algorithm terminates and produces a diagram which is equivalent in meaning to the original, we show that linearising conjuncts reduces the number of non-linear ∧-labelled nodes in a diagram.

Lemma 4.2.1. Let D_1 be a generalized diagram that contains a non-linear ∧-labelled node, n, and let D_2 be the diagram obtained by using rule 4, reduce out-degree of conjunct, to linearise n in D_1. Then the sum of the out-degree of non-linear ∧-labelled nodes in D_2 is less than that of D_1.

Proof. Let S_1 be the sum of the out-degree of non-linear ∧-labelled nodes in D_1, and let $\mathcal{N}_1$ be the set of immediate descendants of n in D_1. Then n has an out-degree of $|\mathcal{N}_1|$ in D_1. Let S_2 be the sum of the out-degree of non-linear ∧-labelled nodes in D_2 and let $\mathcal{N}_2$ be the set of immediate descendants of n in D_2. By the definition of the rule, n has one fewer immediate descendant in D_2 than in D_1: $|\mathcal{N}_2| = |\mathcal{N}_1| - 1$. The ∧-labelled connectives which appear in D_2 but not in D_1 are those introduced by the use of transformation 16, attach diagram to leaf. By the definition of transformation 16, each of these nodes is linear, and so $S_2 = S_1 - 1$.

Thus, the sum of the out-degrees of non-linear $\wedge$-labelled nodes in D_2 is less than that of D_1. $\square$

Theorem 4.2.2: Let D_1 be a generalized diagram. Then we can use algorithm 1 to produce D_2 in finitely many steps, where $D_1 \equiv_{\models} D_2$ and D_2 is in $\wedge$-linear normal form.

Proof. Each step of the algorithm reduces the out-degree of a non-linear $\wedge$-labelled node, n, in D_1, giving a diagram, say D_1'. By lemma 4.2.1, the the sum of the out-degree of the $\wedge$-labelled nodes in D_1' is less than that in D_1, and so the repeated use of the rule results in a diagram, D_2, that contains no non-linear $\wedge$-labelled nodes. By theorem 4.2.1, $D_1 \equiv_{\models} D_2$. Since the sum of the out-degrees of the $\wedge$-labelled nodes in D_1 is finite and each step reduces the out-degree of a non-linear $\wedge$-labelled node, the process terminates. $\square$

4.3 Pushing syntax

In this section we define a series of rules that allow us to add syntax in sound ways to a unitary diagram, d, in the context of a generalized diagram, D, and the syntax of other unitary diagrams labelling nodes in D. The objective is to transform a generalized diagram so that all of its information is represented in leaf nodes, a process which we call ‘pushing’ syntax. Figure 4.35 shows a generalized diagram, D_1, which consists of two nodes labelled by unitary diagrams, d_1 and d_2, in which spiders are labelled x and y for convenience. In order to transform D_1 into a diagram in which all information is represented in leaf nodes we need to push forward the syntax of d_1 into d_2. That is, we need to add the contour labelled A, the spider labelled x and the arrow (f, x, A) to d_2. Several of these operations result in disjunctions; when we push forward the contour labelled A, we need to represent the possibility that y is in A and the possibility that it is not. When we push forward the spider x from d_1 to d_2, we can read from d_1 that x is outside A, but need to represent the possibilities that x is inside or outside of B, and that x is equal to, or distinct from, y.

In Figure 4.36, diagram D_2 is the result of two operations on D_1: prefixing D_1 with a root node labelled by a unitary diagram that contains all of the contours of D_1, and pushing all syntax in D_1.

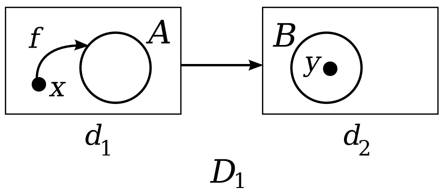


Figure 4.35: A generalized diagram prior to pushing syntax.

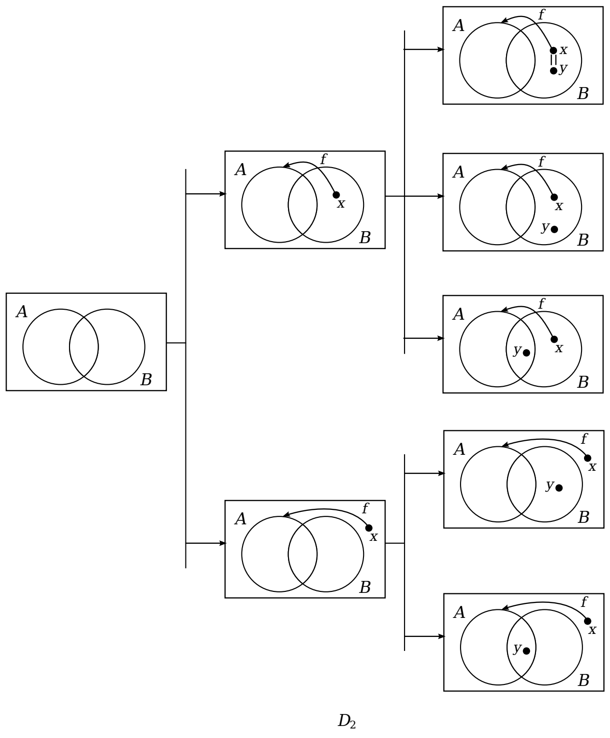


Figure 4.36: A generalized diagram after pushing syntax.

As a second example, we return to the running example used throughout this chapter. In section 4.2, we transformed this diagram to $\wedge$-linear normal form, as shown in Figure 4.37. Before pushing syntax, we prefix the diagram with a root node labelled by a unitary diagram that contains all of the contours of D_2, as shown in Figure 4.38, diagram D_3.

This step is not, in fact, necessary in this example, but in general we wish to collect all contours at the root of a diagram before pushing syntax. This is because we want each unitary diagram in a given branch to have the same zone set, and collecting the contours at the root of a diagram allows us to push the

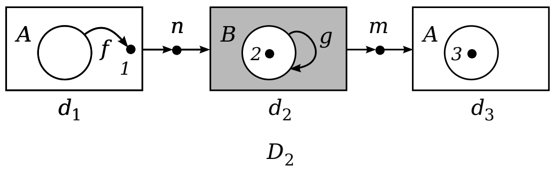


Figure 4.37: A running example: $\wedge$-linear normal form.

same set of contours into every branch. Although it is only necessary that unitary diagrams in the same branch have the same zone set, we push all contours into each disjunctive branch for convenience. The result is that every unitary diagram in the resulting generalized diagram has the same zone set.

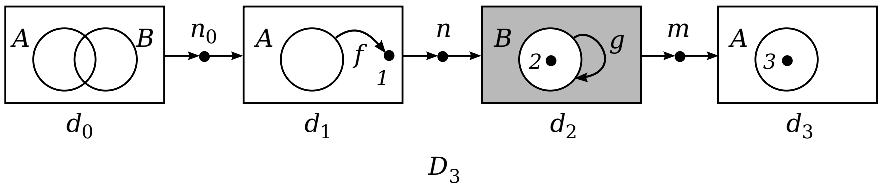


Figure 4.38: A running example: prior to pushing syntax.

Pushing the first contour, B, one step forward from the root node of diagram D_3, Figure 4.38, results in diagram D_4, Figure 4.39. In diagram D_4 the unitary diagram d_{11} and d_{12} reflect the fact that, in diagram D_3, the spider labelled 1 may or may not include B in its habitat. The nodes labelled m^+ and the unitary diagrams labelled d_2^+ and d_3^+ are copies of their counterparts from D_3. In the case of the unitary diagrams d_2^+ and d_3^+, the nodes they label will have been renamed so as to be unique in D_4.

Figure 4.40 places this section in the larger context of the decision procedure, where the outer dashed box indicates the current stage. In this stage, a diagram in $\wedge$-linear normal form, D_2, is transformed to a diagram in pushed syntax normal form, D_3. The inner dashed box illustrates the fact that the transformation to pushed syntax normal form is a two-stage process. First, we collect a full set of the contours of the original diagram, D_2, at the root of the diagram, giving D_3'. This part of the process is described in section 4.3.1. Then we transform D_3' to D_3, which is in pushed syntax normal form, by pushing syntax towards the leaves, as described in section 4.3.2.

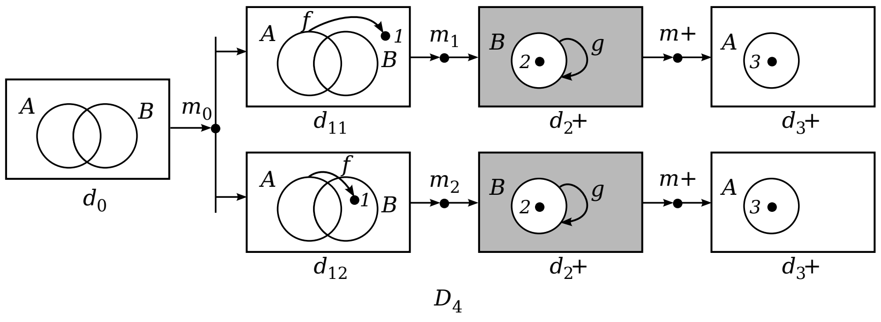


Figure 4.39: A running example: after pushing the first contour one step.

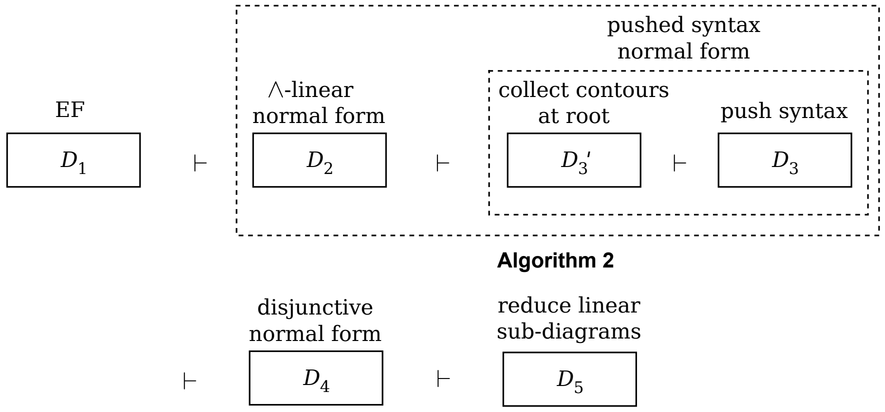


Figure 4.40: Pushed syntax normal form in the context of the decision procedure.

4.3.1 Preparations for pushing syntax

Before pushing syntax, we want unitary diagrams labelling nodes within a particular branch to have the same set of contours and zones. This condition is ensured by the stronger, and conceptually simpler, condition that each unitary diagram has the same set of zones. There is no guarantee that we can achieve the latter condition simply by pushing syntax forwards, since a contour may appear in one branch but not another. In Figure 4.41, diagram D_1, for example, we cannot obtain a diagram with equal zone sets in its unitary diagrams simply by pushing syntax. Instead, we prepare the diagram by collecting a full set of contours at its root. In Figure 4.41, diagram D_2, we have attached D_1 to a diagram that consists of a single node labelled by the trivial diagram.

In Figure 4.42 we have collected the contours of D_2, Figure 4.41, and added

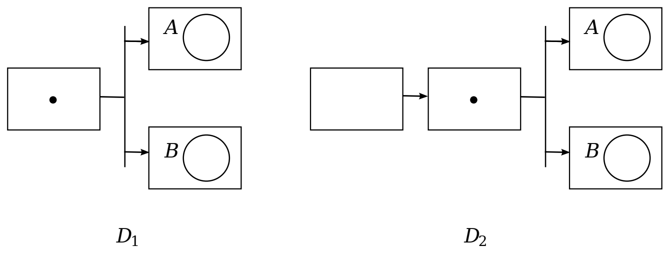


Figure 4.41: Preparing a diagram before pushing syntax, part 1.

them to the root node, producing D_3. Since we add the contours so that they split every zone, we are making no assertions about the intersection or disjointness of the sets represented, and such an operation will always be valid. So, in the new root node, we have formed a Venn diagram containing all contours in D_3.

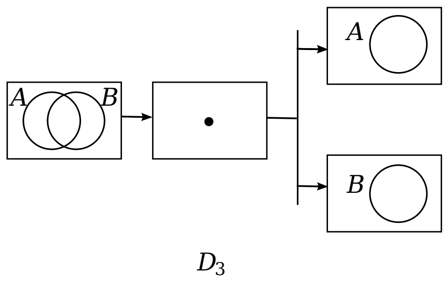


Figure 4.42: Preparing a diagram before pushing syntax, part 2.

Rule 5: Attach to trivial diagram. Let D_1 be a generalized diagram and let $D_\top$ be the generalized diagram that consists of a single node, n, labelled by the trivial diagram. Let D_2 be the diagram obtained by attaching the root node of D_1 to n in $D_\top$ using $\wedge$ and transformation 16, attach diagram to leaf: $D_\top \xrightarrow{+(n,\wedge,D_1)} D_2$. Then D_1 can be replaced by D_2 and vice versa.

Rule 6: Add contour. We now define the rule which, in the context of a generalized diagram, adds a contour to a unitary diagram which contains no spiders, arrows or shading.

Formal description. Let D_1 be a generalized diagram that includes a node, n, labelled by a unitary diagram, d_1, where d_1 contains no spiders, shading or arrows. Let $c \in \mathcal{C} - C(d_1)$ and let d_2 be the unitary diagram obtained by adding

c to d_1 using transformation 12, add contour, as follows:

$$d_1 \xrightarrow{+P} d_2,$$

where $P = (Z_{in}, Z_{out}, S_{in}, S_{out})$ satisfies the following:

1. $Z_{in} = Z_{out} = C(d_1)$, and
2. $S_{in} = S_{out} = \emptyset$.

Let D_2 be the diagram obtained by using transformation 18, relabel node, to relabel n by d_2 in D_1: $D_1 \xrightarrow{l(n,d_2)} D_2$. Then D_1 can be replaced by D_2 and vice versa.

We can use the add contour rule to collect a full set of the contours of a generalized diagram, D, at its root. The final preparation required before pushing syntax in D is to transform each unitary diagram in D to a Venn diagram. We do this by adding missing zones. First, we define a rule which adds a missing zone to a unitary diagram. Recall the definition of missing zones in definition 2.2.6, page 38.

Rule 7: Add missing zone. We can add a missing, shaded zone to a diagram by using the add zone transformation. Figure 4.43 shows the addition of the missing zone $(\{A, B\}, \{C\})$ to d_1 to give d_2.

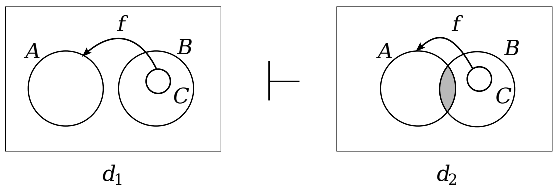


Figure 4.43: An application of *add missing zone*.

Formal description. Let d_1 ($\neq \perp$) be a generalized unitary diagram and z be a zone such that $z \in MZ(d_1)$. Let d_2 be the diagram obtained by composing transformation 11, add zone, and transformation 10, add shading, to add z as a new shaded zone: $d_1 \xrightarrow{+z} d_1' \xrightarrow{+z^*} d_2$. Then we can replace d_1 by d_2.

Rule 8: Add missing zone (generalized). In order to apply the previous rule in the context of a generalized diagram, we need to combine that (unitary-to-unitary) inference rule with the transformation which relabels a node in a generalized diagram, and now define an inference rule that does so.

Formal description. Let D be a generalized diagram which contains a node, n, labelled by a unitary diagram, d_1, and let $z \in MZ(d_1)$. Let d_2 be the unitary diagram obtained by using the add missing zone rule to add z to d_1. Let D_2 be the generalized diagram obtained by relabelling n by d_2 in D_1: $D_1 \xrightarrow{\rho(n,d_2)} D_2$. Then D_1 can be replaced by D_2 and vice versa.

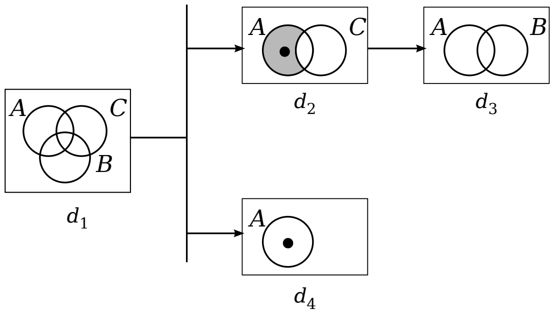


Figure 4.44: A generalized diagram in which we wish to push syntax forwards.

After pushing contours from the root to the leaves of a diagram, we want each unitary diagram to have not only the same set of contours but the same set of zones. We achieve this by converting each unitary diagram to its equivalent Venn form (defined on page38), and by pushing contours so Venn form is maintained.

Definition 4.3.1. Let $D_1 = (V_1, V_2, E, l)$ be a generalized diagram. We define the **Venn form** of D_1, denoted $Venn(D_1)$, to be the diagram, D_2, which is a copy of D_1 in which each diagram-labelled node is relabelled by the Venn form of its label in D_1.

4.3.2 Inference rules which push syntax

In order to push syntax, we need to push diagrammatic elements in the right order. In Figure 4.44, we cannot push the shading in the zone $(\{A\}, \{C\})$ from d_2 to d_3 because that zone does not exist in d_3. Before we can push the shading in d_2 forwards, we need to push the contours forwards from the root, equalising the zone sets of the unitary diagrams. Pushing contours becomes the first step in the process of pushing syntax.

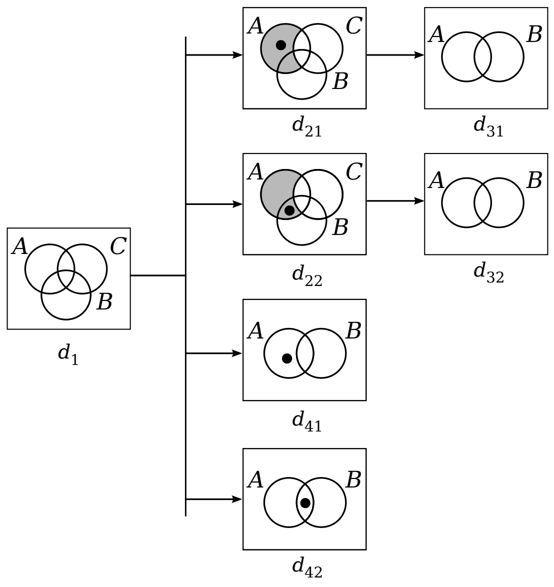


Figure 4.45: Pushing the first contour forwards.

Figure 4.45 shows the result of pushing B forwards one level from the root in Figure 4.44. The unitary diagram d_2 from Figure 4.44 is replaced by the disjunction of two new unitary diagrams, d_{21} and d_{22}. Similarly, d_4 is replaced by the disjunction of d_{41} and d_{42}. If we continue this process to push all contours as far as the leaves, the result is the diagram in Figure 4.46. In Figure 4.46, each unitary diagram has the same zone set. Thus, the habitat of any spider is present in each unitary diagram, and we are ready to push spiders forward. The result of pushing spiders, followed by pushing shading, is shown in Figure 4.47. If there were any arrows to be pushed forwards, this could be done at any time after pushing contours and spiders is completed, by which point the source and target of each arrow would necessarily be present in each unitary diagram.

As we have described, we will push contours before any other syntax. However, we will define the rules in a different order, beginning with the simplest cases, those rules which push shading and arrows.

Rule 9: Push shading. If a zone z is shaded in a unitary diagram d_1 and is unshaded in d_2, the immediate diagram-labelled descendant of d_1, where d_1 and

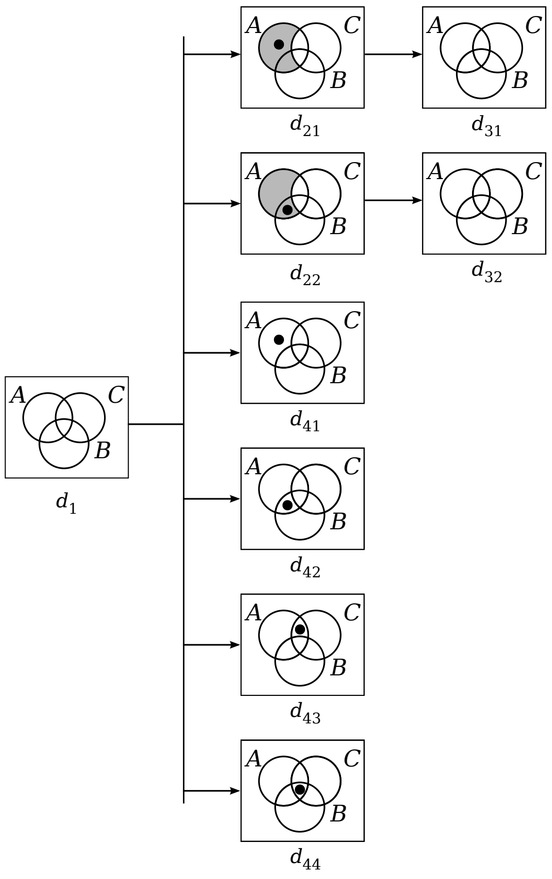


Figure 4.46: Pushing contours forwards.

d_2 have the same zone sets and the spiders of d_2 are a superset of or equal to the spiders of d_1, then we can add shading to z in d_2. Figure 4.48 shows an example. In diagram D_1 the zone $(\emptyset, \{A\})$ is shaded in d_1 but not in its descendant, d_2. In d_2, $(\emptyset, \{A\})$ contains a superset of the spiders inhabiting the same zone in d_1. Pushing the shading forward to d_2 results in diagram D_2.

Formal description. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams d_1 and d_2 respectively, such that

1. n_1 is an immediate diagram-labelled ancestor of n_2,
2. $Z(d_1) = Z(d_2)$,
3. $S(d_1) \subseteq S(d_2)$, and

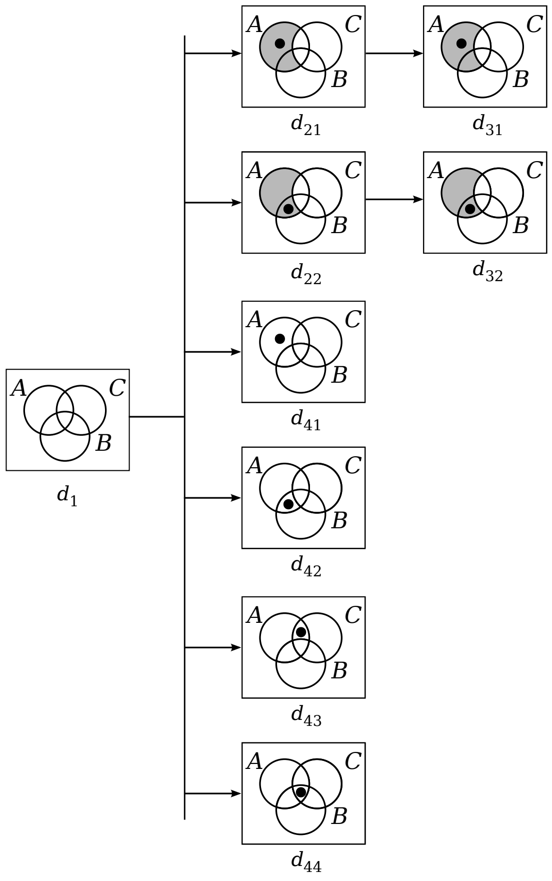


Figure 4.47: Pushing other syntax forwards.

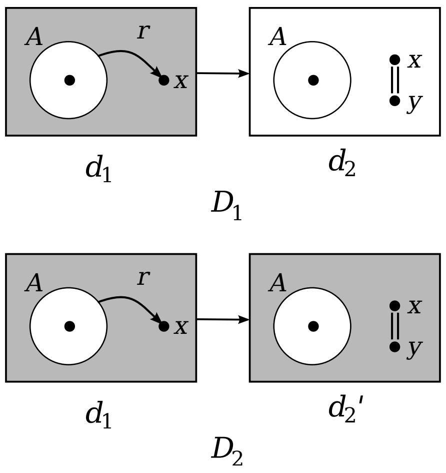


Figure 4.48: Pushing shading from one unitary diagram to another.

4. there exists a zone, z, which is shaded in d_1 and unshaded in d_2: $z \in Z^*(d_1) - Z^*(d_2)$.

Let d_3 be the diagram obtained by adding shading to z to d_2: $d_2 \xrightarrow{+z^*} d_3$. Let D_2 be a copy of D_1 in which n_2 is relabelled by d_3 using transformation 18, relabel node: $D_1 \xrightarrow{l(n_2,d_3)} D_2$. Then D_1 can be replaced with D_2 and vice versa.

The next rule we require is one which pushes arrows forwards from one unitary diagram to another.

Rule 10: Push arrow. Arrows are pushed forward after other types of syntax. Thus, since contours and spiders have been pushed forward, the sources and targets of all arrows are necessarily present. Figure 4.49 shows an example: in D_1 the arrow (r, A, x) is present in d_1 but not d_2. The source and target of (r, A, x) are present in d_2 and so we are able to add the arrow, resulting in D_2.

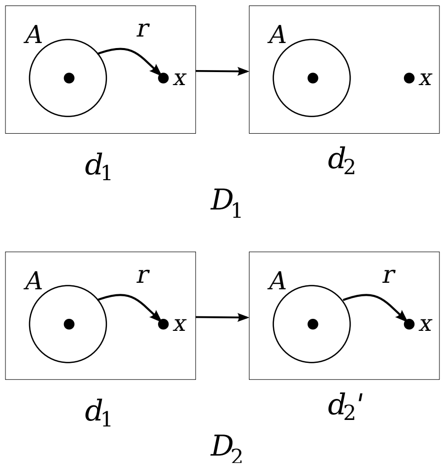


Figure 4.49: Pushing an arrow from one unitary diagram to another.

Formal description. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams d_1 and d_2 respectively, such that

1. n_1 is the immediate diagram-labelled ancestor of n_2,

2. $Z(d_1) = Z(d_2)$,

3. $S(d_1) \subseteq S(d_2)$, and

4. there exists an arrow, (l,s,t), which is present in d_1 and not in d_2: $(l,s,t) \in A(d_1) - A(d_2)$.

Let d_3 be the diagram obtained by adding (l,s,t) to d_2: $d_2 \xrightarrow{+(l,s,t)} d_3$. Let D_2 be a copy of D_1 in which n_2 is labelled by d_3 using transformation 18, relabel node: $D_1 \xrightarrow{l(n_2,d_3)} D_2$. Then D_1 can be replaced with D_2 and vice versa.

Note that conditions (2) and (3) of the previous rule, $Z(d_1) = Z(d_2)$ and $S(d_1) \subseteq S(d_2)$, imply that the source and target of the arrow to be pushed forwards are present in d_2.

We now move on to define rules in which the act of pushing syntax forwards may result in a disjunction representing various possibilities for the habitat and identity of spiders. As described in the example illustrated by Figure 4.44, pushing a contour results in a disjunction representing the possible habitats of spiders. We now define the unitary diagrams that make up this disjunction.

Definition 4.3.2. Let d_1 be a generalized unitary diagram and let $c \in \mathcal{C} - C(d_1)$. Let d_2 be a diagram obtained under transformation 12, add contour: $d_1 \xrightarrow{+P} d_2$, where P is of the form $(c, Z_{in}, Z_{out}, S_{in}, S_{out})$ and satisfies the following:

1. $Z_{in} = Z_{out} = Z(d_1)$,
2. $S_{in} \subseteq S(d_1)$,
3. $S_{out} = S(d_1) - S_{in}$.

Then we say that d_2 is an **add contour component** of d_1 for c. We denote the set of all such unitary diagrams by $ACC(c, d_1)$.

In Figure 4.50, diagrams d_2 and d_3 are the add contour components of d_1 for the contour C.

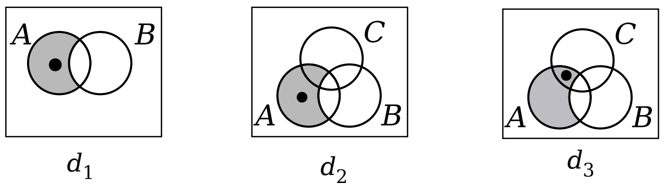


Figure 4.50: Add contour components.

When making use of add contour components in inference rules, we will require them to be generalized diagrams with a single node, rather than unitary diagrams. Thus, we define the add contour component diagrams.

Definition 4.3.3. Let d_1 be a generalized unitary diagram, let $c \in \mathcal{C} - C(d_1)$, let d_2 be an add contour component of d_1 for c and let $n \in \mathcal{V}$. Let $D_2 = (V, W, E, l)$ be the generalized diagram with a single node, n, labelled by d_2. That is, D_2 satisfies

1. $V = \{n\}$,
2. $W = \emptyset$,
3. $E = \emptyset$, and
4. $l = \{(n, d_2)\}$,

Then we say that D_2 is an **add contour component diagram** of d_1 for c. We denote a minimal set of add contour component diagrams of d_1 for c by $ACCD(c, d_1)$, where each element of $ACC(c, d_1)$ labels a unique diagram in $ACCD(c, d_1)$.

We will define the rule which pushes contours in the context of an initial diagram in $\wedge$-linear normal form. The following example illustrates the reason for this constraint: in Figure 4.51, the unitary diagrams d_{21} and d_{22} are the add contour components for d_2 in diagram D_1, with respect to the unlabelled derived contour in unitary diagram d_1. Thus, to push forward the derived contour from d_1 to d_2, we want to replace the node labelled by d_2 with the disjunction of nodes labelled by d_{21} and d_{22}. Given that the immediate ancestor of the node labelled by d_2 is a non-linear $\wedge$-labelled node, there is no clear way to do this, without making complex changes to the structure of D_1.

In Figure 4.52, diagram D_2 has nodes labelled by the same unitary diagrams as in diagram D_1, Figure 4.51, but the non-linear node is labelled by $\vee$ rather than $\wedge$. To push forward the derived contour from d_1 to d_2, the add contour components are the same as in the previous case, diagrams d_{21} and d_{22}, Figure 4.51. In this example, the node labelled by d_2 can be replaced by the disjunction of the add contour components in a straightforward way, resulting in diagram D_3. It is for this reason that we define the rule which pushes contours in the context of an

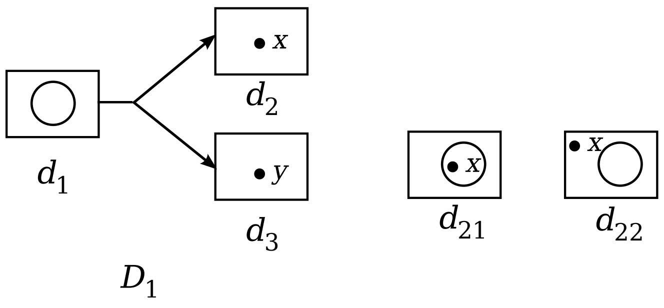


Figure 4.51: Pushing contours without ∧-linear normal form.

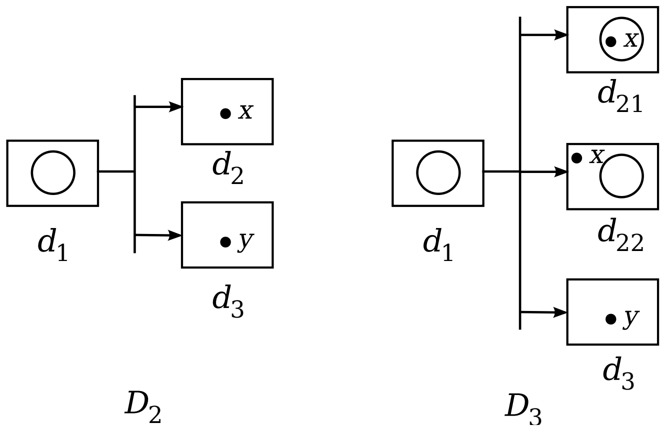


Figure 4.52: Pushing contours with ∧-linear normal form.

initial diagram in ∧-linear normal form. We will apply the same constraint to the rules which push spiders, as they may also produce a disjunction.

Rule 11: Push contour. We now define the rule which pushes contours forwards in a diagram which is in ∧-linear normal form.
Formal description. Let D_1 be a generalized diagram in ∧-linear normal form, containing two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams d_1 and d_2 respectively, such that n_2 is an immediate diagram-labelled descendant of n_1. Let c be a contour in d_1 but not d_2: $c \in C(d_1) - C(d_2)$. Let D_2 be the diagram obtained by replacing n_2 by the disjunction of the set of diagrams $ACCD(c, d_2)$ in D_1 using transformation 23, replace inner node: $D_1 \xrightarrow{\rho(n_2,\vee,ACCD(c,d_2))} D_2$. Then D_1 can be replaced by D_2 and vice versa.

Next, we define two rules which push spiders forwards between unitary diagrams having the same set of zones. Pushing forward a spider x from a unitary diagram d_1 to its immediate diagram-labelled descendant d_2 gives rise to a disjunction representing the possibilities of the identity of x with regard to the spiders sharing its habitat in d_2. As we will see, the possibilities differ according to whether the habitat of x is shaded in d_2. We deal first with the case of pushing a spider forward into a shaded zone.

When pushing a spider, x, into a shaded zone, we know that the zone contains exactly as many elements as are represented by spiders. We therefore need to represent the possibilities for the equality of x with each of the spiders in the zone of the unitary diagram to which x will be pushed. In Figure 4.53, there are two spiders in d_1, labelled x and y. In d_2, there are also two spiders inhabiting A, x and z. The spider y is omitted from d_2 and its habitat in d_1 is shaded in d_2.

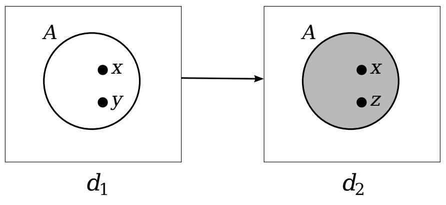


Figure 4.53: Preparing to push a spider into an shaded zone.

Figure 4.54 shows the result of pushing y forward from d_1 to d_2 in Figure 4.53. We know from the shading in the zone $(\{A\}, \emptyset)$ in d_2, Figure 4.53, that the number of distinct spiders in that zone cannot increase without changing the meaning of d_2, unless new spiders are joined by a tie to existing spiders. Therefore, pushing y forwards gives the disjunction of d_{21} and d_{22}, expressing the fact that y may be equal to x or z (but not both, since we know that $x \neq z$ from diagram d_2, Figure 4.53).

Similarly to the definition of add contour components, we define the set of unitary diagrams that make up the possibilities for adding a spider with ties to the spiders in a particular zone. In Figure 4.54, d_{21} and d_{22} are the *add spider with ties* components of the spider y in Figure 4.53, diagram d_1.

Definition 4.3.4. Let d_1 be a generalized unitary diagram, let $x \in \mathcal{S} - S(d_1)$, let $z \in Z(d_1)$ and let $y \in S(z, d_1)$. Let d_2 be the diagram obtained by using the add spider with ties inference rule to add x to d_1 with habitat z and with a tie

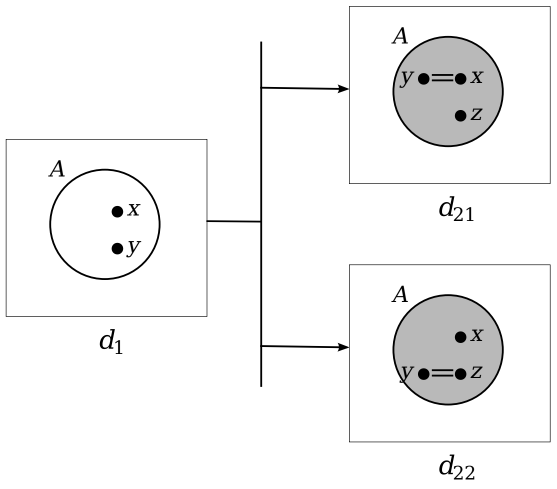


Figure 4.54: The result of pushing a spider into an shaded zone.

joining x and y. Then we say that d_2 is an **add spider with ties component** of d_1 for x and z. We denote the set of add spider with ties components for all spiders inhabiting z by $ASTC(x, z, d_1)$.

Definition 4.3.5. Let d_1 be a generalized unitary diagram, let $x \in \mathcal{S} - S(d_1)$, let $z \in Z(d_1)$, let $y \in S(z, d_1)$ and let d_2 be an add spider with ties component of d_1 for x and z. Let $n \in \mathcal{V}$ and let $D_2 = (V, W, E, l)$ be the generalized diagram with a single node, n, labelled by d_2. That is, D_2 satisfies

1. $V = \{n\}$,
2. $W = \emptyset$,
3. $E = \emptyset$, and
4. $l = \{(n, d_2)\}$,

Then we say that D_2 is an **add spider with ties component diagram** of d_1 for x and z. We denote a minimal set of add spider with ties component diagrams of d_1 for c by $ASTCD(x, z, d_1)$, where each element of $ASTC(x, z, d_1)$ labels a unique diagram in $ASTCD(x, z, d_1)$.

Rule 12: Push spider (shaded habitat). Given two unitary diagrams, d_1 and d_2, and a spider, x, in $S(d_1) - S(d_2)$, there is a case in which there are no add spider with ties components for x. This is illustrated by Figure 4.55. There are no add spider with ties components of d_2 for the spider labelled x. We can see that d_1 and d_2 are mutually inconsistent, as they make incompatible assertions about the cardinality of the set represented by A. A rule which pushed x from d_1 to be d_2 would be unsound, and so we exclude this case in the following definition.

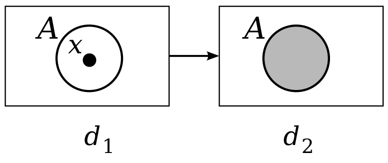


Figure 4.55: Inconsistency due to spiders and shading.

Formal description. Let D_1 be a generalized diagram in $\wedge$-linear normal form, containing two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams, d_1 and d_2, respectively, such that:

1. n_1 is the immediate diagram-labelled ancestor of n_2,
2. $Z(d_1) = Z(d_2)$,
3. there exists a spider, x, which is present in d_1 and missing from d_2: $x \in S(d_1) - S(d_2)$,
4. the habitat of x is shaded in d_2: $\eta_{d_1}(x) \in Z^*(d_2)$, and
5. the habitat of x contains one or more spiders in d_2: $S(\eta_{d_1}(x), d_2) \neq \emptyset$.

Let $\mathcal{D} = ASTCD(x, \eta_{d_1}(x), d_2)$ and let D_2 be the diagram obtained by using transformation 23, replace inner node, to replace n_2 by the disjunction of the diagrams in $\mathcal{D}$ in D_1: $D_1 \xrightarrow{\rho(n_2,\vee,\mathcal{D})} D_2$. Then D_1 can be replaced by D_2 and vice versa.

As we saw in the example illustrated by Figure 4.55, if conditions (1) to (4) of the above rule are met, but condition (5) is not met, then the diagram in question is inconsistent. When we encounter such a situation, we want to 'record' it, in some sense, to make use of the information in later steps. We do this by taking a diagram such as that shown in Figure 4.55, relabelling the first node in the pair that gives rise to the inconsistency (the root node, in this example) by $\perp$, the unitary diagram that represents falsity, and then removing the descendants of the node.

Rule 13: Replace inconsistent pair. We now define an inference rule which replaces diagrams such as that shown in Figure 4.55 with $\perp$.
Formal description. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams, d_1 and d_2, respectively, such that:

1. n_2 is an immediate diagram-labelled descendant of n_1,
2. $Z(d_1) = Z(d_2)$,
3. there exists a spider, x, which is present in d_1 and missing in d_2: $x \in S(d_1) - S(d_2)$,
4. the habitat of x is shaded in d_2: $\eta_{d_1}(x) \in Z^*(d_2)$, and
5. the habitat of x contains no spiders in d_2: $S(\eta_{d_1}(x), d_2) = \emptyset$.

Let $\mathcal{X}$ be the set of sub-diagrams of D_1 induced by the immediate diagram-labelled descendants of n_2 and let D_1' be the diagram obtained by using transformation 13, remove sub-diagram, to remove all members of $\mathcal{X}$ from D_1:

$$D_1 \xrightarrow{-X_1} D_{X_1} \ldots D_{X(n-1)} \xrightarrow{-X_n} D_1',$$

for $\mathcal{X} = \{X_1 \ldots X_n\}$. Let D_2 be the diagram obtained by using transformation 18, relabel node, to relabel n_2 by $\perp$ in D_1': $D_1' \xrightarrow{l(n_2,\perp)} D_2$. Then D_1 can be replaced by D_2 and vice versa.

We now turn to pushing spiders into unshaded zones. Unlike the previous case, this cannot give rise to inconsistencies.

Rule 14: Push spider (unshaded habitat). In Figure 4.56 the spider labelled x in d_1 is not present in d_2. In order to push x from d_1 to d_2 we need to include the case where x represents the same element as does the unlabelled spider inhabiting the zone $(\{A\}, \emptyset)$ in d_2, and the case where it does not.

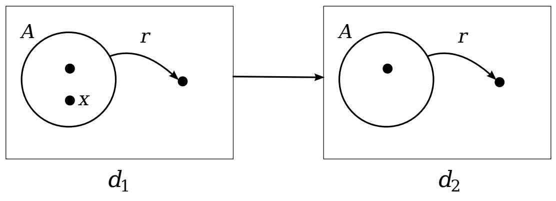


Figure 4.56: Preparing to push a spider into an unshaded zone.

Figure 4.57 shows the result of pushing the spider, x, from d_1 to d_2 in Figure 4.56. The possibilities for the equality or inequality of x with the spiders sharing its habitat in d_2 are represented by the disjunction of the diagrams d_{21} and d_{22}. These diagrams comprise the add spider with ties components for x and the zone $(\{A\}, \emptyset)$ and one extra diagram, in which x represents a distinct element.

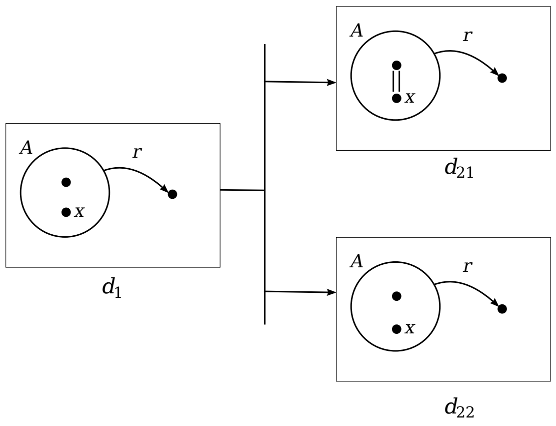


Figure 4.57: The result of pushing a spider into an unshaded zone.

Formal description. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2 labelled by generalized unitary diagrams d_1 and d_2 respectively such that

1. n_1 is the immediate diagram-labelled ancestor of n_2,
2. $Z(d_1) = Z(d_2)$,
3. there exists a spider, x, which is present in d_1 and missing in d_2: $x \in S(d_1) - S(d_2)$, and

4. the habitat of x is unshaded in d_2: $\eta_{d_1}(x) \notin Z^*(d_2)$.

Let d_x be the diagram obtained by adding x to d_2 with habitat $\eta_{d_1}(x)$: $d_2 \xrightarrow{+(x,\eta_{d_1}(x))} d_x$. Let $\mathcal{D}$ be the set of diagrams formed as follows:

$$\mathcal{D} = \{d_x\} \cup ASTCD(x, \eta_{d_1}(x), d_2).$$

Let D_2 be the diagram obtained by using transformation 23, replace inner node, to replace n_2 by the disjunction of the diagrams in $\mathcal{D}$ in D_1: $D_1 \xrightarrow{\rho(n_2,\vee,\mathcal{D})} D_2$. Then D_1 can be replaced by D_2 and vice versa.

4.3.3 Validity of the inference rules which push syntax

In this section we show that the rules required by the process of pushing syntax are sound.

Theorem 4.3.1: Rule 5, attach to trivial diagram, is valid. Let D_1 be a generalized diagram and let $D_\top$ be the generalized diagram that consists of a single node, n, labelled by the trivial diagram. Let D_2 be the diagram obtained by using the rule to attach the root node of D_1 to n in $D_\top$. Then $D_1 \equiv_\vDash D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. The formula for D_2 is equal to the formula for $D_\top$ conjoined with that for D_1: $\mathit{form}(D_2) = \mathit{form}(D_\top) \wedge \mathit{form}(D_1)$. The formula for $D_\top$ is always true, and so the formula for D_2 is true whenever the formula for D_1 is true. Thus, if I' is valid for D_1, it is also valid for D_2 and $D_1 \vDash D_2$. We can see that $D_2 \vDash D_1$ by a similar argument, so $D_1 \equiv_\vDash D_2$. □

Theorem 4.3.2: Rule 6, add contour, is valid. Let D_1 be a generalized diagram that includes a node, n, labelled by a unitary diagram, d_1, where d_1 contains no spiders, shading or arrows. Let $c \in \mathcal{C} - C(d_1)$. Let D_2 be the diagram obtained by using the rule to relabel n by a unitary diagram that contains c. Then $D_1 \equiv_\vDash D_2$.

Proof. As in the definition of the rule, let d_2 be the unitary diagram obtained by adding c to d_1 using transformation 12, add contour, as follows:

$$d_1 \xrightarrow{+P} d_2,$$

where $P = (Z_{in}, Z_{out}, S_{in}, S_{out})$ satisfies the following:

1. $Z_{in} = Z_{out} = C(d_1)$, and

2. $S_{in} = S_{out} = \emptyset$.

Let $I = (U, \Psi, \Phi)$ be a model for d_1 with valid extension $I' = (U, \Psi', \Phi)$. In [38, p74], in the proof of the corresponding add contour rule for SD2, it is shown that $\Psi'(Z(d_1)) = \Psi'(Z(d_2))$. Therefore, the plane tiling condition holds for d_2 under I'. Since d_1, and therefore d_2, contain no spiders, shading or arrows, the other conditions hold trivially and $\mathit{form}(d_2)$ is true under I'. Diagram D_2 is a copy of D_1 in which d_1 is replaced by d_2, so it follows that I' is valid for D_2 and $D_1 \vDash D_2$. Conversely, the result in [38, p74] states that the plane tiling condition is true for d_1 whenever it is true for d_2, so we can show that $D_2 \vDash D_1$ by a similar argument. Therefore, $D_1 \equiv_{\vDash} D_2$ as required. □

Before showing that the rule which adds missing zones to a diagram is valid, we state the result which shows that missing zones represent the empty set.

Lemma 4.3.1. Let d be a generalized unitary diagram and let $I = (U, \Psi, \Phi)$ be a model for d with valid extension $I' = (U, \Psi', \Phi)$. Let $z \in MZ(d)$. Then $\Psi'(z) = \emptyset$.

Proof. By the plane tiling condition for d, $\Psi'(Z(d)) = U$. Since distinct zones represent disjoint sets in U, $\Psi'(z) = \emptyset$. □

Theorem 4.3.3: Rule 7, add missing zone, is valid. Let d_1 $(\neq \bot)$ be a generalized unitary diagram and z be a zone such that $z \in MZ(d_1)$. Let d_2 be the diagram obtained by using the rule to add z to d_1. Then $d_1 \equiv_{\vDash} d_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be a model for d_1 with valid extension $I' = (U, \Psi', \Phi)$. Since z is missing from d_1 and by lemma 4.3.1, $\Psi'(z) = \emptyset$. The diagram d_2 is a copy of d_1 which includes z as an untouched, shaded zone. Thus, the plane tiling condition holds for d_2 under I'. We know that $S(z, d_2) = \emptyset$, so the shaded zones condition holds since

$$\Psi'(z) = \Big(\bigcup_{x \in S(z,d_2)} \Psi'(x) \Big) = \emptyset.$$

The other conditions hold trivially, and so I' is valid for d_2 and $d_1 \vDash d_2$. We can show that $d_2 \vDash d_1$ by a similar argument, and so $d_1 \equiv_{\vDash} d_2$. □

Theorem 4.3.4: Rule 8, add missing zone (generalized), is valid. Let D be a generalized diagram which contains a node, n, labelled by a unitary diagram, d_1 ($\neq \bot$), and let $z \in MZ(d)$. Let D_2 be the generalized diagram obtained by using the rule to relabel n by a unitary diagram that contains z. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be a model for D_1 with valid extension $I' = (U, \Psi', \Phi)$. As in the definition of the rule, let d_2 be the unitary diagram obtained by using the add missing zone rule to add z to d_1. The formula for D_2 is equivalent to a copy of the formula for D_1 in which the sub-formula $\mathit{form}(d_1)$ is replaced by $\mathit{form}(d_2)$. By the soundness of the rule 7, add missing zone, $d_1 \models d_2$ and so $\mathit{form}(d_2)$ is true under I' if $\mathit{form}(d_1)$ is true under I'. It follows that I' is valid for D_2 and $D_1 \models D_2$. By theorem 4.3.3, it is also true that $d_2 \models d_1$ and so, by the same argument, $D_2 \models D_1$ and $D_1 \equiv_{\models} D_2$ as required. □

Recall the definition of the Venn form of generalized diagrams, given in definitions 2.2.6 and 4.3.1. We can produce the Venn form of a generalized diagram, D, by repeated application of rule 8, add missing zone (generalized). By the soundness of this rule, the Venn form of D has an equivalent meaning to D.

Corollary 4.3.1. Let D be a generalized diagram. Then $D \equiv_{\models} \mathit{Venn}(D)$.

Proof. Each diagram-labelled node in $\mathit{Venn}(D)$ is labelled by a unitary diagram, say $\hat{d}$, which is the Venn form of the diagram labelling the corresponding node in D, say d. So, $\hat{d} = \mathit{Venn}(d)$. $\mathit{Venn}(d)$ can be produced by repeated applications of rule 7, add missing zone, and by theorem 4.3.3, $d \equiv_{\models} \mathit{Venn}(d)$. Since the same is true for each diagram-labelled node, it follows that $D \equiv_{\models} \mathit{Venn}(D)$. □

Next, we show that the rules which push syntax are sound.

Theorem 4.3.5: Rule 9, push shading, is valid. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2 labelled by generalized unitary diagrams d_1 and d_2 respectively such that

1. n_1 is the immediate diagram-labelled ancestor of n_2,
2. $Z(d_1) = Z(d_2)$,
3. $S(d_1) \subseteq S(d_2)$, and

4. there exists a zone, z, which is shaded in d_1 and unshaded in d_2: $z \in Z(d_1) - Z(d_2)$.

Let d_3 be the diagram obtained by adding shading to z in d_2: $d_2 \xrightarrow{+z^*} d_3$. Let D_2 be the diagram obtained by using the rule to relabel n_2 in D_1 by d_3. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We will show that I' is valid for D_1 if and only if I' is valid for D_2. First, we show that if I' is valid for d_1 and d_2, then I' is valid for d_3. Now, the semantic formula for d_3 is equivalent to that for d_2 conjoined with the shaded zones condition for z:

$$\mathit{form}(d_3) \equiv \mathit{form}(d_2) \wedge \big(\Psi'(z) = \bigcup_{x \in S(z,d_2)} \Psi'(x)\big). \tag{4.2}$$

Trivially, we see that to establish the truth of $\mathit{form}(d_3)$, we must show $\Psi'(z) = \bigcup\limits_{x \in S(z,d_2)} \Psi'(x)$. Now, by the shaded zones condition for d_1,

$$\Psi'(z) = \bigcup_{x \in S(z,d_1)} \Psi'(x). \tag{4.3}$$

By the spiders habitat condition for d_2,

$$\bigcup_{x \in S(z,d_2)} \Psi'(x) \subseteq \Psi'(z). \tag{4.4}$$

Since $S(z, d_2) = S(z, d_3)$ we have, by (4.4),

$$\bigcup_{x \in S(z,d_3)} \Psi'(x) \subseteq \Psi'(z). \tag{4.5}$$

Suppose there exists an element, e, where

$$e \in \Psi'(z) - \bigcup_{x \in S(z,d_3)} \Psi'(x). \tag{4.6}$$

That is, (4.5) is, in fact, a proper subset relationship. By (4.3), $\{e\}$ is mapped to a spider, say x, in d_1: $\Psi'(x) = \{e\}$. Since $x \in S(d_1)$, we further know that

$x \in S(d_2)$, since $S(d_1) \subseteq S(d_2)$. By the spiders habitat condition for d_2,

$$\Psi'(x) \subseteq \Psi'(\eta_{d_2}(x)).$$

From this, and since distinct zones represent disjoint sets, it follows that $\eta_{d_2}(x) = z$. Now $\eta_{d_2}(x) = \eta_{d_3}(x) = z$, contradicting (4.6). Hence,

$$\bigcup_{x \in S(z,d_3)} \Psi'(x) = \Psi'(z),$$

as required. Hence, *form*(d_3) is true and I' is valid for d_3. Diagram D_2 is a copy of D_1 in which d_2 is replaced by d_3, and so I' is valid for D_2 if I' is valid for D_1.

Next, we show the reverse. By (4.2), we can see that *form*(d_3) implies *form*(d_2), so if *form*(d_1) and *form*(d_3) are true under I', then *form*(d_2) is true under I'. That is, I' is valid for d_2 and, since D_1 is a copy of D_2 in which d_3 is replaced by d_2, I' is valid for D_1 if I' is valid for D_2. Therefore, $D_1 \equiv_{\models} D_2$ as required. □

Theorem 4.3.6: Rule 10, push arrow, is valid. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2 labelled by generalized unitary diagrams d_1 and d_2 respectively such that

1. n_2 is an immediate diagram-labelled descendant of n_1,
2. $Z(d_1) = Z(d_2)$,
3. $S(d_1) \subseteq S(d_2)$, and
4. there exists an arrow, (l, s, t), which is present in d_1 and not in d_2: $(l, s, t) \in A(d_1) - A(d_2)$.

Let d_3 be the diagram obtained by adding (l, s, t) to d_2, or $d_2 \xrightarrow{+(l,s,t)} d_3$ and let D_2 be the diagram obtained by using the rule to relabel n_2 by d_3 in D_1. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be a model for D_1 with valid extension $I' = (U, \Psi', \Phi)$. We will show that I models D_2, and in particular that I' is valid for D_2. Consider the semantic formula for D_2. Now, the semantic formula for d_3 is equivalent to that of d_2 conjoined with $\Psi'(s).\Phi(l) = \Psi'(t)$:

$$form(d_3) = form(d_2) \wedge \Psi'(s).\Phi(l) = \Psi'(t). \tag{4.7}$$

Moreover,

$$form(d_1) \wedge (form(d_2) \dots) \tag{4.8}$$

is equivalent to a sub-formula of $form(D_1)$, so by (4.7),

$$form(d_1) \wedge ((form(d_2) \wedge \Psi'(s).\Phi(l) = \Psi'(t)) \dots) \tag{4.9}$$

is a sub-formula of $form(D_2)$. If (4.8) is true, we know $\Psi'(s).\Phi(l) = \Psi'(t)$ since $(l, s, t) \in A(d_1)$ and by the arrows condition for d_1. So, if (4.8) is true then so is (4.9). Clearly it follows then that I' is valid for D_2 and $D_1 \vDash D_2$. D_2 is a copy of D_1 with one extra arrows, and so the argument which shows $D_2 \vDash D_1$ is straightforward. Hence, $D_1 \equiv_{\vDash} D_2$. □

Theorem 4.3.7: Rule 11, push contour, is valid. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams d_1 and d_2 respectively, such that n_1 is the immediate diagram-labelled ancestor of n_2. Let c be a contour in d_1 but not d_2: $c \in C(d_1) - C(d_2)$. Let D_2 be the diagram obtained by using the rule to add c to d_2. Then $D_1 \equiv_{\vDash} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We will show that I' is valid for D_1 if and only if I' is valid for D_2. First, we show that if I' is valid for d_1 and d_2, then I' is valid for one of the unitary diagrams in the set $ACC(c, d_2)$, the add contour components for c in d_2. First, we show that the plane tiling condition is true for each $d_i \in ACC(c, d_2)$. By the definition of add contour components, c is added to d_2 so that it splits every zone, or $Z_{in} = Z_{out} = Z(d_2)$. Thus, by the definition of the add contour transformation, for each zone $(in, out) \in Z(d_2)$, the zones $(in \cup \{c\}, out)$ and $(in, out \cup \{c\})$ are in $Z(d_i)$. By corollary 2.3.1,

$$\begin{aligned}\Psi'(in, out) &= \Psi'(in \cup \{c\}, out) \cup \Psi'(in, out \cup \{c\}) \\ &= \Psi'((in \cup \{c\}, out), (in, out \cup \{c\})).\end{aligned}$$

Thus we have

$$\begin{aligned}\Psi'(Z(d_2)) &= \bigcup_{(in,out)\in Z(d_2)} \Psi'(in, out)\\ &= \bigcup_{(in,out)\in Z(d_2)} \Psi'((in \cup \{c\}, out), (in, out \cup \{c\}))\\ &= \Psi'(Z(d_i)). \qquad (4.10)\end{aligned}$$

Therefore the plane tiling condition is true for d_i under I' if it is true for d_2. Next, we show that the remaining conditions hold for some member of $ACC(c, d_2)$, beginning with the spiders habitat condition. Let S_{in} and S_{out} be defined as follows:

$$S_{in} = \{x \in S(d_2) : \Psi'(x) \subseteq \Psi'(in \cup c, out)\}, \qquad (4.11)$$

and

$$S_{out} = \{x \in S(d_2) : \Psi'(x) \subseteq \Psi'(in, out \cup c)\}. \qquad (4.12)$$

By the spiders' distinctness condition for d_2, S_{in} and S_{out} partition $S(d_2)$. By the definition of add contour components, we know that there is a unitary diagram in $ACC(c, d_2)$ where c was added to d_2 using S_{in} and S_{out} to parametrise the add contour rule. Let d_i be such a unitary diagram. By the spiders habitat condition for d_2, and by (4.11) and (4.12), the spiders habitat condition is true for d_i under I' if it is true for d_2.

We now show that the shaded zones condition holds for d_i. Let $(in, out) \in Z^*(d_i)$. Either $c \in in$ or $c \in out$. Assume $c \in in$. In this case,

$$S((in, out), d_i) = \{x \in S_{in} : x \in S((in - \{c\}, out), d_2)\}.$$

Let e be an element in $\Psi'(in, out)$. Since $c \in in$ and by the definition of Ψ' for zones, $e \in \Psi'(c)$ and $e \in \Psi'(in - \{c\}, out)$. By the definition the rule, we know that the zone $(in - \{c\}, out)$ is shaded in d_2 since (in, out) is shaded in d_i. By the shaded zones condition for d_2, e is represented by a spider, say x, in d_2. By (4.11), $x \in S_{in}$, so $x \in S((in, out), d_i)$. Since this is true for any e, we have

$$\Psi'(in, out) = \bigcup_{x\in S((in,out),d_i)} \Psi'(x).$$

Thus the shaded zones condition is true for (in, out) in d_i under I' if $c \in in$. If c is instead in out, then the reasoning is similar. Therefore the shaded zones condition is true for d_i under I' if it is true for d_2. Now, the spiders' distinctness and arrows conditions are unaffected since d_i contains the same spiders and arrows as d_2. Thus, both conditions hold for d_i under I' if they hold for d_2, and so $form(d_2)$ implies $form(d_i)$. Thus,

$$form(d_2) \Rightarrow \bigvee_{d_i \in ACC(c,d_2)} form(d_i).$$

The formula for D_2 is equivalent to a copy of the formula for D_1 in which $form(d_2)$ is replaced by the above disjunction. So we can see that the formula for D_2 is true under I' whenever the formula for D_1 is true, and $D_1 \vDash D_2$.

To show the reverse, we will show that the formula for any $d_i \in ACC(c, d_2)$ implies the formula for d_2. Let $d_i \in ACC(c, d_2)$. By (4.10), the plane tiling condition is true for d_2 under I' when it is true for d_i. To show the shaded zones condition, let $(in, out) \in Z^*(d_2)$ and assume that the shaded zones condition is true for d_i under I'. By the definition of the rule, the zones $(in \cup \{c\}, out)$ and $(in, out \cup \{c\})$ are shaded in d_i. Then

$$\Psi'(in \cup \{c\}, out) = \bigcup_{x \in S((in \cup \{c\}, out), d_i)} \Psi'(x), \tag{4.13}$$

and

$$\Psi'(in, out \cup \{c\}) = \bigcup_{x \in S((in, out \cup \{c\}), d_i)} \Psi'(x). \tag{4.14}$$

By corollary 2.3.1,

$$\Psi'(in, out) = \Psi'(in \cup \{c\}, out) \cup \Psi'(in, out \cup \{c\}), \tag{4.15}$$

which we can rewrite by (4.13) and (4.14):

$$\Psi'(in, out) = \big(\bigcup_{x \in S((in \cup \{c\}, out), d_i)} \Psi'(x)\big) \cup \big(\bigcup_{x \in S((in, out \cup \{c\}), d_i)} \Psi'(x)\big). \tag{4.16}$$

By the definition of the rule, each spider $x \in S((in, out), d_2)$ is in $S(d_i)$ and either $x \in S_{in}$ or $x \in S_{out}$. If $x \in S_{in}$, then $x \in S((in \cup \{c\}, out), d_i)$, and if

$x \in S_{out}$, then $x \in S((in, out \cup \{c\}), d_i)$. Therefore,

$$S((in, out), d_2) \subseteq S((in \cup \{c\}, out), d_i) \cup S((in, out \cup \{c\}), d_i). \tag{4.17}$$

We know that S_{in} and S_{out} partition the spiders of d_i, and thus they also partition the spiders of d_2. Therefore, if $x \in S_{in}$ or $x \in S_{out}$, then x is in $S(d_2)$, so

$$S((in \cup \{c\}, out), d_i) \cup S((in, out \cup \{c\}), d_i) \subseteq S((in, out), d_2).$$

By this and (4.17),

$$S((in, out), d_2) = S((in \cup \{c\}, out), d_i) \cup S((in, out \cup \{c\}), d_i).$$

By this and the shaded zones condition for d_i,

$$\Psi'(in, out) = \bigcup_{x \in S((in,out),d_2)} \Psi'(x),$$

and the shaded zones condition is true for d_2 under I' if it is true for d_i.

To show the spiders habitat condition, let $x \in S(d_2)$ and let $(in, out) = \eta_{d_2}(x)$. Assume that the spiders habitat condition is true for d_i under I'. We will show that $\Psi'(x) \subseteq \Psi'(in, out)$. We know that $x \in S(d_i)$ and either $x \in S_{in}$, in which case $x \in S((in \cup \{c\}, out), d_i)$, or $x \in S_{out}$, in which case $x \in S((in, out \cup \{c\}), d_i)$. Therefore,

$$\begin{aligned} \Psi'(x) &\subseteq \Psi'(in \cup \{c\}, out) \cup \Psi'(in, out \cup \{c\}) && \text{by } SHC(d_i) \\ &\subseteq \Psi'(in, out) && \text{by (4.15).} \end{aligned}$$

Thus, the spiders habitat condition is true for d_2 under I' if it is true for d_i. Again, the spiders distinctness and arrows conditions are unaffected, so both conditions hold for d_2 under I' if they hold for d_2, and so $form(d_i)$ implies $form(d_2)$. It follows that

$$\Big(\bigvee_{d_i \in ACC(c,d_2)} form(d_i)\Big) \Rightarrow form(d_2).$$

Similarly to before, the formula for D_1 is equivalent to a copy of the formula for D_2 in which the above disjunction is replaced by $form(d_2)$. Thus, the formula for D_1 is true under I' whenever the formula for D_2 is true, and $D_2 \vDash D_1$ and $D_1 \equiv_{\vDash} D_2$ as required. □

Theorem 4.3.8: Rule 12, push spider (shaded habitat), is valid. Let D_1 be a generalized diagram in $\wedge$-linear normal form, containing two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams, d_1 and d_2, respectively, such that:

1. n_1 is the immediate diagram-labelled ancestor of n_2,
2. $Z(d_1) = Z(d_2)$,
3. there exists a spider, x, which is present in d_1 and missing from d_2: $x \in S(d_1) - S(d_2)$,
4. the habitat of x is shaded in d_2: $\eta_{d_1}(x) \in Z^*(d_2)$, and
5. the habitat of x contains one or more spiders in d_2: $S(\eta_{d_1}(x), d_2) \neq \emptyset$.

Let $\mathcal{D} = ASTCD(x, \eta_{d_1}(x), d_2)$ and let D_2 be the diagram obtained by using the rule to replace the node n_2 in D_1 by the disjunction of $\mathcal{D}$. Then $D_1 \equiv_{\vDash} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We will show that I' is valid for D_1 if and only if I' is valid for D_2. First, we show that if I' is valid for d_1 and d_2, then I' is valid for one of the unitary diagrams in the set $ASTC(x, \eta_{d_1}(x), d_2)$, the add spider with ties components for x in d_2. The plane tiling condition holds for each $d_i \in ASTC(x, \eta_{d_1}(x), d_2)$ trivially, if it is true for d_1 and d_2. By condition (5), the habitat of x contains one or more spiders in d_2. Thus, the shaded zones condition is true for each $d_i \in ASTC(x, \eta_{d_1}(x), d_2)$ if it is true for d_2, since x is joined by a tie to a spider in the zone which it is placed, and the number of distinct spiders is not increased. The spiders habitat condition is true for each $d_i \in ASTC(x, \eta_{d_1}(x), d_2)$ if it is true in d_1, since x is placed in the zone $\eta_{d_1}(x)$. To show the spiders' distinctness condition, choose a $d_i \in ASTC(x, \eta_{d_1}(x), d_2)$ such that x is tied to exactly those spiders in d_2 with which it is equal:

$$(x, y) \in \tau_{d_i} \Leftrightarrow \Psi'(x) = \Psi'(y), \tag{4.18}$$

for $y \in S(\eta_{d_1}(x), d_2)$. It follows that the spiders' distinctness condition is true for d_i under I' if it is true for d_2. By the definition of add spider with ties components, such a d_i exists. Thus, $\mathit{form}(d_1)$ and $\mathit{form}(d_2)$ imply $\mathit{form}(d_i)$, and

$$\big(\mathit{form}(d_1) \wedge \mathit{form}(d_2)\big) \Rightarrow \bigvee_{d_i \in ASTC(x,\eta_{d_1}(x),d_2)} \mathit{form}(d_i).$$

The formula for D_2 is equivalent to a copy of the formula for D_1 in which $\mathit{form}(d_2)$ is replaced by the above disjunction. So we can see that the formula for D_2 is true under I' whenever the formula for D_1 is true, and $D_1 \vDash D_2$.

We can see that, for each $d_i \in ASTC(x, \eta_{d_1}(x), d_2)$, $\mathit{form}(d_i)$ implies $\mathit{form}(d_2)$, since $\mathit{form}(d_2)$ is a sub-formula of $\mathit{form}(d_i)$. Therefore,

$$\Big(\bigvee_{d_i \in ASTC(x,\eta_{d_1}(x),d_2)} \mathit{form}(d_i)\Big) \Rightarrow \mathit{form}(d_2).$$

As before, it follows from this that the formula for D_1 is true under I' whenever the formula for D_2 is true, $D_2 \vDash D_1$ and $D_1 \equiv_{\vDash} D_2$ as required. □

Theorem 4.3.9: Rule 13, replace inconsistent pair, is valid. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2, labelled by generalized unitary diagrams, d_1 and d_2, respectively, such that:

1. n_2 is an immediate diagram-labelled descendant of n_1,
2. $Z(d_1) = Z(d_2)$,
3. there exists a spider, x, which is present in d_1 and missing in d_2: $x \in S(d_1) - S(d_2)$,
4. the habitat of x is shaded in d_2: $\eta_{d_1}(x) \in Z^*(d_2)$, and
5. the habitat of x contains no spiders in d_2: $S(\eta_{d_1}(x), d_2) = \emptyset$.

Let D_2 be the diagram obtained by using the rule to remove the descendants of n_2 from D_1 and to relabel n_2 by $\perp$. Then $D_1 \equiv_{\vDash} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. If the spiders habitat condition is true for d_1 then $\Psi'(\eta_{d_1}(x)) \neq \emptyset$. However, $S(\eta_{d_1}(x), d_2) = \emptyset$, and since $\eta_{d_1}(x) \in Z^*(d_2)$, $\Psi'(\eta_{d_1}(x)) = \emptyset$. Therefore, if the spiders habitat condition is true for d_1, then the shaded zones condition is false for d_2. By a similar argument, if the shaded zones condition is true for d_2 then the spiders habitat condition is false for d_1. Thus, $\mathit{form}(d_1)$ and $\mathit{form}(d_2)$ cannot be true at the same time. Let D_{n_2} be the sub-diagram of D_1 induced by n_2. Let $\mathcal{X}$ be the set of sub-diagrams of D_1 induced by the immediate diagram-labelled descendants of n_1 and suppose that $n_\diamond$, the immediate descendant of n_1, is labelled by $\vee$. Then the formula for D_1 contains the following subformula:

$$\mathit{form}(d_1) \wedge \bigvee_{X_i \in \mathcal{X}} \mathit{form}(X_i). \tag{4.19}$$

We know that $D_{n_2} \in \mathcal{X}$ and that $\mathit{form}(D_{n_2})$ is prefixed by $\mathit{form}(d_2)$. Since $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ evaluates to $\perp$, $\mathit{form}(d_1) \wedge \mathit{form}(D_{n_2})$ evaluates to bot. Thus, (4.19) is equivalent to

$$\mathit{form}(d_1) \wedge \bigvee_{X_i \in (\mathcal{X} - D_{n_2}) \cup \{\perp\}} \mathit{form}(X_i). \tag{4.20}$$

The formula for D_2 is a copy of the formula for D_1 in which (4.19) is replaced by (4.20). Thus, if $n_\diamond$, the immediate descendant of n_1, is labelled by $\vee$, then D_1 is satisfiable if and only if D_2 is satisfiable. If $n_\diamond$ is labelled by $\wedge$, the result follows by a similar argument. Therefore, $D_1 \equiv_\models D_2$ as required. □

Theorem 4.3.10: Rule 14, push spider (unshaded habitat), is valid. Let D_1 be a generalized diagram which includes two diagram-labelled nodes, n_1 and n_2 labelled by generalized unitary diagrams, d_1 and d_2, respectively such that

1. n_1 is the immediate diagram-labelled ancestor of n_2,
2. $Z(d_1) = Z(d_2)$,
3. there exists a spider, x, which is present in d_1 and missing in d_2: $x \in S(d_1) - S(d_2)$, and
4. the habitat of x is unshaded in d_2: $\eta_{d_1}(x) \notin Z^*(d_2)$.

Let d_x be the diagram obtained by adding x to d_2 with habitat $\eta_{d_1}(x)$: $d_2 \xrightarrow{+(x,\eta_{d_1}(x))} d_x$. Let $\mathcal{D}$ be the set of diagrams formed as follows:

$$\mathcal{D} = \{d_x\} \cup ASTCD(x, \eta_{d_1}(x), d_1).$$

Let D_2 be the diagram obtained by using the rule to replace the node n_2 by the disjunction of the diagrams in $\mathcal{D}$. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be a model for D_1 with valid extension $I' = (U, \Psi', \Phi)$. We will show that I models D_2, and in particular that I' is valid for D_2. If $\Psi'(x)$ is represented by a spider in d_2, the proof is the same as that of theorem 4.3.8. Suppose that this is not the case. That is, $\Psi'(x) \neq \Psi'(y)$ for all $y \in S(\eta_{d_1}(x), d_2)$. The formula for d_x is equal to the formula for d_2 conjoined with the two statements $\Psi'(x) \in \Psi'(\eta_{d_x}(x))$ (the spiders' habitat condition for d_x) and $\Psi'(x) \neq \Psi'(y)$ for all spiders $y \in S(\eta_{d_1}(x), d_2)$ (the spiders distinctness condition for d_x). We know that $\eta_{d_x}(x) = \eta_{d_1}(x)$ and that $\Psi'(x) \in \Psi'(\eta_{d_1}(x))$ by the spiders' habitat condition for d_1, and so the spiders habitat condition is true for d_x under I' if it is true in d_1. We know that $x \notin S(d_2)$ and have assumed that x is not represented by any of the spiders of d_2, so the spiders distinctness condition holds for d_x under I' if it holds for d_2. The other conditions are trivially true, and so we have the following:

$$\big(\mathit{form}(d_1) \wedge \mathit{form}(d_2)\big) \Rightarrow \mathit{form}(d_x).$$

Since $d_x \in \mathcal{D}$,

$$\big(\mathit{form}(d_1) \wedge \mathit{form}(d_2)\big) \Rightarrow \bigvee_{d' \in \mathcal{D}} \mathit{form}(d'). \tag{4.21}$$

The formula for D_2 is equivalent to a copy of the formula for D_1 in which $\mathit{form}(d_2)$ is replaced by the above disjunction, and so I' is valid for D_2 and $D_1 \models D_2$. The steps required to show that $D_2 \models D_1$ are the same as those in the proof of theorem 4.3.8, and so $D_1 \equiv_{\models} D_2$ as required. □

4.3.4 Pushed syntax normal form

We are now able to define the normal form possessed by a diagram in which all syntax has been pushed to the leaves, and show that an initial diagram can be transformed into an equivalent diagram that has this form. In Figure 4.58, diagram D_7 is the diagram obtained by pushing all syntax to the leaves in our running example. So, diagram D_7 is in pushed syntax normal form. Note that this results in several inconsistencies: in the unitary diagram d_{32}, the spider labelled 3 inhabits a zone outside of B, something which its immediate diagram-labelled ancestor tells us cannot be true. Such inconsistencies are detected and removed in the following section.

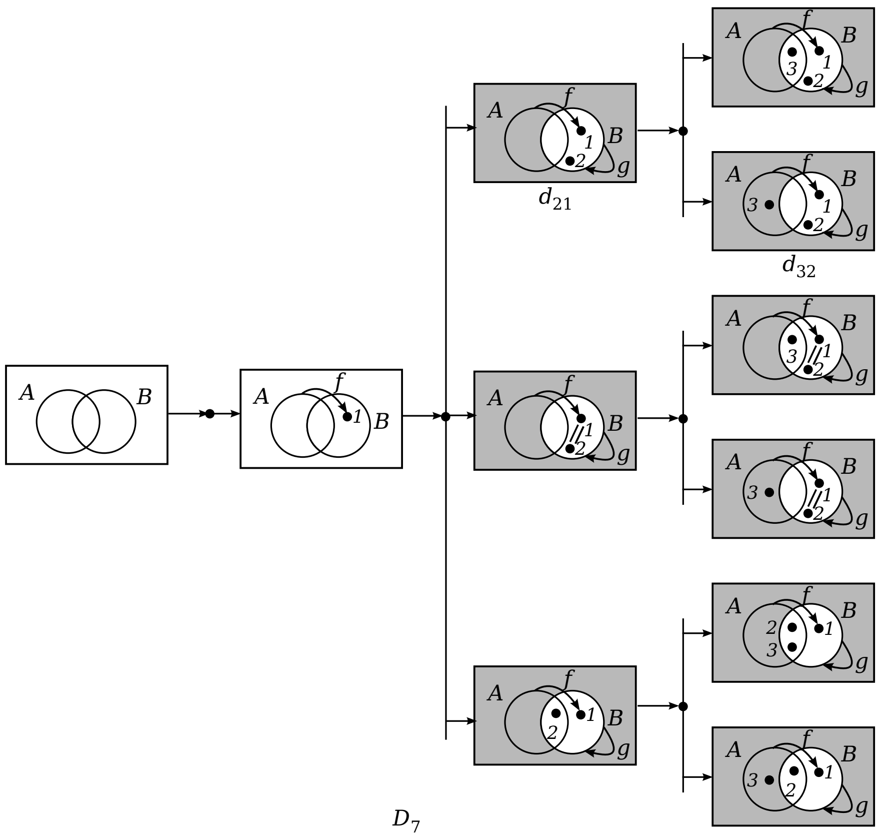


Figure 4.58: A running example: pushed syntax normal form.

Pushed syntax normal form will be defined in terms of an ordering over the syntax of a generalized diagram. Given two unitary diagrams, d_1 and d_2, d_1 is a *syntactic sub-diagram* of d_2 if and only if all of the syntax of d_1 appears within d_2. In Figure 4.59, d_1 is a syntactic sub-diagram of d_2 and it is easy to see that we can remove syntax from d_2 to produce d_1.

Definition 4.3.6. Let d_1 and d_2 be generalized unitary diagrams. We say that d_1 is a **syntactic sub-diagram** of d_2, denoted $d_1 \subseteq_S d_2$, if and only if the following is true:

1. $Z(d_1) = Z(d_2)$,
2. $Z^*(d_1) \subseteq Z^*(d_2)$,

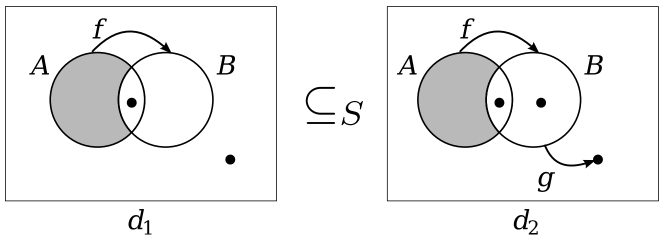


Figure 4.59: The syntactic sub-diagram relationship.

3. $S(d_1) \subseteq S(d_2)$,
4. $A(d_1) \subseteq A(d_2)$.

Recall that a diagram in $\wedge$-linear normal form is one in which all $\wedge$-labelled nodes are linear. We build on this normal form in the next definition.

Definition 4.3.7. Let D be a generalized diagram in $\wedge$-linear formal form. If, for each pair of diagram-labelled nodes n_1 and n_2, labelled by unitary diagrams d_1 and d_2, where n_1 is the immediate diagram-labelled ancestor of n_2, it is the case that either $d_1 \subseteq_S d_2$ or $d_2 = \perp$, then we say that D is in **pushed syntax normal form**.

Next we state the algorithm used to transform a diagram in $\wedge$-linear normal form to one in pushed syntax normal form.

Algorithm 2 ($\wedge$-linear normal form to pushed syntax normal form)**.** Let D_1 be a generalized diagram in $\wedge$-linear normal form which is not in pushed syntax normal form and where each diagram-labelled node, n_i, in D_1 is labelled by a unitary diagram named d_i. Transform D_1 into a diagram, D_2, using the following steps.

1. Use rule 5, attach to trivial diagram, to attach D_1 to the trivial generalized diagram, obtaining E_1. Let $\mathcal{C}$ be the (finite) contour set of E_1.
2. Use repeated applications of rule 6, add contour (contours only), to add each element of $\mathcal{C}$ to the root node of E_1, obtaining E_2.
3. Use rule 8, add missing zone (generalized), to add missing zones to each unitary diagram labelling a node in E_2 until each unitary diagram is transformed to a Venn diagram, calling the resulting generalized diagram E_3.

4. Next, push the contours of E_3 towards its leaves, as described below, starting with the root node, say n_0. If n_0 is a leaf node then we are done. Otherwise, pick an immediate diagram-labelled descendant, say n_1. We compare the contours of d_0 to those of d_1. If there exists a contour, c, in $C(d_0) - C(d_1)$, we use rule 11, push contour, to push c from d_0 to d_1, and continue in this way until $C(d_0) = C(d_1)$. Next, we push contours from d_0 to the unitary diagram labelling the next immediate diagram-labelled descendant of n_0, if any. We carry on in this way until all contours have been pushed to each unitary diagram in E_3, obtaining diagram E_4, in which each unitary diagram has the same zone set.

5. Next, push the spiders of E_4 towards its leaves. Recall that the root node of E_4 contains no spiders; therefore, choose one of the immediate diagram-labelled descendants of n_0, say n_1, and an immediate diagram-labelled descendant of n_1, say n_2, and begin to push spiders from d_1 to d_2. Let x be a spider in $S(d_1) - S(d_2)$ and assume the habitat of x is shaded in d_2. In order for rule 12, push spider (shaded habitat), to be applicable, we require the following to be true:

   (a) $Z(d_1) = Z(d_2)$,

   (b) x is present in d_1 and missing in d_2,

   (c) the habitat of x is shaded in d_2, and

   (d) the habitat of x contains one or more spiders in d_2.

   Conditions 5a to 5c are true since we have equalised the zone sets of the unitary diagrams in E_4 in step (4), and by assumption. If condition 5d is true, use rule 12 to push x from d_1 to d_2, giving diagram E_4'. If condition 5d is false, use rule 13, replace inconsistent pair, to relabel n_1 by $\perp$ and remove its descendants from E_4, giving diagram E_4'.

   Otherwise, the habitat of x is not shaded in d_2. Then we use rule 14, push spider (unshaded habitat) to push x from d_1 to d_2. Note that the conditions of the rule are met since $Z(d_1) = Z(d_2)$. As with the contours, we carry on this process until, for each branch, all spiders have been pushed as far as the leaf or the branch has a leaf node labelled by $\perp$. Let E_5 be the resulting generalized diagram.

6. Next, we use rule 9, push shading, to push the shading in zones from each unitary diagram in E_5 to its immediate diagram-labelled descendants. Now, during the previous step we may have introduced $\perp$ as the label of leaf nodes. Therefore we know that, for each unitary diagram d_i and immediate diagram-labelled descendant d_j, $Z(d_i) = Z(d_j)$ and $S(d_i) \subseteq S(d_j)$, or $d_j = \perp$. If $d_j = \perp$, we do not push shading. If $d_j \neq \perp$, the conditions for the use of rule 9 to push the shading from each zone which is shaded in d_i but not d_j are met, and we do so. We continue this process for each branch until we reach the leaf and call the resulting diagram E_6.

7. Finally, we use rule 10, push arrow, to push arrows from each unitary diagram to its immediate diagram-labelled descendants. As with shading, we know that, for each unitary diagram d_i and immediate diagram-labelled descendant d_j, either $Z(d_i) = Z(d_j)$ and $S(d_i) \subseteq S(d_j)$, or $d_j = \perp$. If $d_j = \perp$, we do not push arrows. If $d_j \neq \perp$, the conditions for the use of rule 10 to push arrows from d_i to d_j are met, and we do so. We continue this process until we reach the leaves and call the resulting diagram D_2.

Next we show that the above algorithm is terminating, produces a diagram which is in both $\wedge$-linear normal form and pushed syntax normal form, and which is equivalent to the original diagram.

Theorem 4.3.11: Let D_1 be a generalized diagram in $\wedge$-linear normal form. Then we can use algorithm 2 to produce a diagram, D_2, in finite steps, where D_2 is in $\wedge$-linear normal form and pushed syntax normal form, and where $D_1 \equiv_{\models} D_2$.

Proof. 1. In step (1), diagram E_1 is obtained by using rule 5, attach to trivial diagram, to attach D_1 to the trivial generalized diagram. By theorem 4.3.1, $D_1 \equiv_{\models} E_1$.

2. Step (2) makes use of repeated applications of rule 6, add contour (contours only), to add each element of the finite set $C(E_1)$ to the root node of E_1, obtaining E_2. By theorem 4.3.2, $E_1 \equiv_{\models} E_2$.

3. In step (3) we use rule 8, add missing zone (generalized), to add missing zones to each unitary diagram labelling a node in E_2 until each of those unitary diagrams is transformed to a Venn diagram. Each unitary diagram

has a finite set of missing zones. We call the resulting generalized diagram E_3. We can see that $E_3 = \mathit{Venn}(E_2)$ and, by lemma 4.3.1, $E_2 \equiv_{\models} E_3$.

4. In step (4), the (finite) set of contours of E_3 are pushed towards its leaves, giving diagram E_4. By theorem 4.3.7, $E_3 \equiv_{\models} E_4$.

5. In step (5), the (finite) set of spiders of E_4 are pushed towards its leaves, giving diagram E_5. Let d_1 and d_2 be a unitary diagram labelling a node in E_5 and one of its immediate diagram-labelled descendants, respectively. Let x be a spider in $S(d_1) - S(d_2)$ and assume the habitat of x is shaded in d_2. Then either rule 12, push spider (shaded habitat), or rule 13, replace inconsistent pair, is applicable, giving diagram E_4'. That is, E_4' is a copy of E_4 in which either x is pushed forwards to d_2, or n_1 is relabelled by $\perp$. By theorems 4.3.8 and theorem 4.3.9, $E_4' \equiv_{\models} E_4$. If the habitat of x is not shaded in d_2 then we use rule 14, push spider (unshaded habitat) to push x from d_1 to d_2, giving diagram E_4. By theorem 4.3.10, $E_4' \equiv_{\models} E_4$. This process is repeated until, for each branch, all spiders have been pushed as far as the leaf or the branch has a leaf node labelled by $\perp$. Diagram E_5 is the resulting diagram. By theorems 4.3.8, 4.3.9 and 4.3.10, $E_4 \equiv_{\models} E_5$.

6. In step (6), shading is pushed into diagrams not equal to $\perp$. E_5 has a finite set of shaded zones. We continue this process for each branch of E_5 until we reach the leaf and call the resulting diagram E_6. By theorem 4.3.5, $E_5 \equiv_{\models} E_6$.

7. In step (7), the (finite) set of arrows of E_6 are pushed forwards into unitary diagrams not equal to $\perp$. We continue this process for each branch until we reach the leaf and call the resulting diagram D_2. By theorem 4.3.6, $E_6 \equiv_{\models} D_2$.

Since D_1 is in $\wedge$-linear normal form and since none of the rules used increase the number of non-linear $\wedge$-labelled nodes in the diagram, D_2 is in $\wedge$-linear normal form. Having pushed all types of syntax from the root to the leaves of D_2, it must be true that for all nodes n_i with immediate diagram-labelled descendant n_j, $d_i \subseteq_S d_j$ or $d_j = \perp$. Thus D_2 is in pushed syntax normal form. Since D_1 contains finite syntax, the process is terminating. Finally, since we have constructed a sequence of equivalent diagrams, $D_1 \equiv_{\models} D_2$. □

4.4 Disjunctive normal form

We are now ready to describe our final normal form, which we call *disjunctive normal form*. A generalized diagram in disjunctive normal form is either a linear diagram, or is composed of a root node labelled by the trivial diagram, followed by a disjunctive-labelled node, to which are attached linear sub-diagrams. That is, if a non-linear generalized diagram, D, is in disjunctive normal form, then D has one $\vee$-labelled node and exactly one node, n, labelled by the trivial diagram, which is the ancestor of that node. Furthermore, the sub-diagrams of D induced by the immediate descendants of n are linear sub-diagrams of D. If a diagram has this structure, we say that it has a *trivial prefix*. Figure 4.60 shows the result of transforming our running example, using the rules we will develop in this section, to disjunctive normal form. The diagram in Figure 4.60 has a trivial prefix.

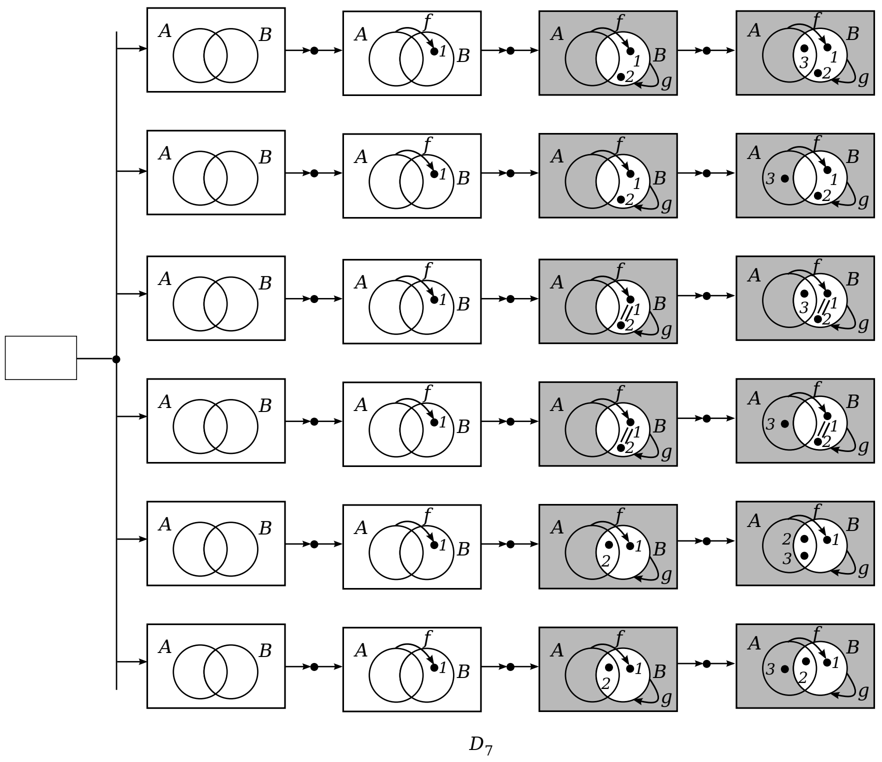


Figure 4.60: A running example: disjunctive normal form.

Definition 4.4.1. Let D be a generalized diagram in $\wedge$-linear normal form. We say that D has a **trivial prefix** if the following conditions are true:

1. the root node, n_1, of D is labelled by $\top$,
2. the immediate descendant of n_1 is a non-linear $\vee$-labelled node, and
3. each of the sub-diagrams of D induced by the immediate diagram-labelled descendants of n_1 is a linear sub-diagram of D.

Our definition of disjunctive normal form does not correspond exactly with the disjunctive normal form found in Boolean logic. In Boolean logic, a formula in disjunctive normal form is one which is a disjunction of conjunctive clauses. The diagram in Figure 4.61 is in disjunctive normal form, and has the following formula:

$$\mathit{form}(d_1) \wedge ((\mathit{form}(d_2) \wedge \mathit{form}(d_3)) \vee (\mathit{form}(d_4) \wedge \mathit{form}(d_5)) \vee \mathit{form}(d_6)),$$

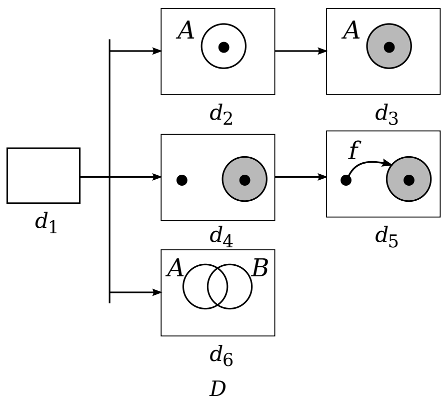


Figure 4.61: Diagrammatic disjunctive normal form.

which is not in disjunctive normal form. The way in which we use the term is the diagrammatic equivalent of the standard sense, however; we require that the only diagram that comes before an (optional) disjunction is labelled by the trivial diagram. Linear diagrams correspond to conjunctive clauses. That is, in Figure 4.61, $\mathit{form}(d_1)$ is trivially true and the meaning conveyed by D is entirely determined by the formula

$$(\mathit{form}(d_2) \wedge \mathit{form}(d_3)) \vee (\mathit{form}(d_4) \wedge \mathit{form}(d_5)) \vee \mathit{form}(d_6)).$$

In general, the meaning of a generalized diagram in disjunctive normal form is determined by a disjunction of conjunctive clauses.

Definition 4.4.2. Let D be a generalized diagram in $\wedge$-linear normal form. We say that D is in **disjunctive normal form** if and only if one of the following conditions is true:

1. D is a linear diagram, or

2. D has a trivial prefix.

Figure 4.62 places this section in the larger context of the decision procedure. The outer dashed box indicates the current stage, in which an original diagram in pushed syntax normal form, D_3, is transformed to a diagram in disjunctive normal form, D_4. The inner dashed box illustrates the fact that the algorithm has two stages. We begin by removing all but one of the non-linear $\vee$-labelled nodes from D_3, giving D_4'. This part of the process is described in section 4.4.1. Next, we transform D_4' into a diagram in disjunctive normal form, D_4, by applying a trivial prefix as described in section 4.4.2.

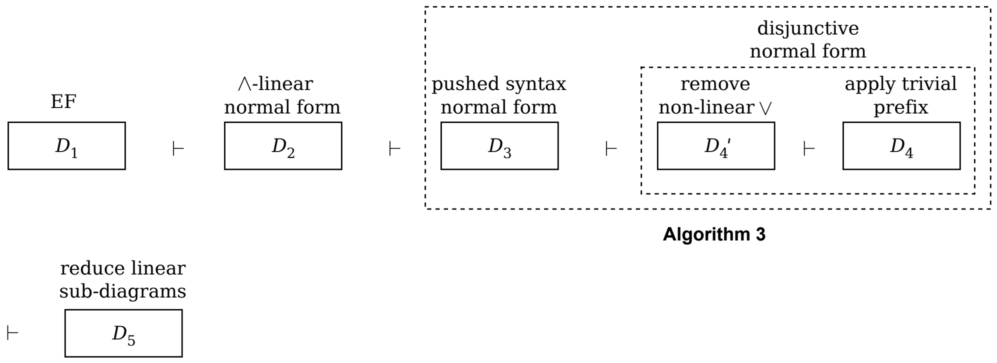


Figure 4.62: Disjunctive normal form in the context of the decision procedure.

The diagrams that we transform to disjunctive normal form are in both $\wedge$-linear and pushed syntax normal forms, and we have shown in previous sections that any diagram can be transformed into an equivalent diagram in these normal forms. When we describe the transformation of a diagram in both $\wedge$-linear and pushed syntax normal forms to disjunctive normal form, we will show that the initial conditions are not affected by the transformation; that is, we will show that the resulting diagram is in $\wedge$-linear, pushed syntax and disjunctive normal forms. We will first consider how to remove non-linear $\vee$-labelled nodes.

4.4.1 Removing non-linear $\vee$-labelled nodes

A non-linear $\vee$-labelled connective, n, in a diagram D, is 'nested' when the path from n to the root of D contains another non-linear $\vee$-labelled node. Also, we say that n is an 'outer' nested non-linear $\vee$-labelled node when n is nested and no non-linear $\vee$-labelled nodes appear in any path from n to the leaves of D.

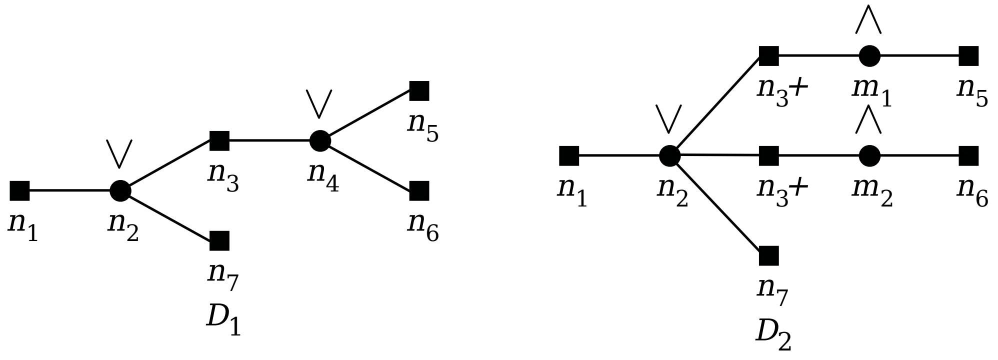


Figure 4.63: Removing a non-linear $\vee$-labelled connective.

Figure 4.63 shows two diagrams, D_1 and D_2. In D_1, n_4 is an outer non-linear $\vee$-labelled node and is nested within n_2. If each diagram-labelled node, n_i, in D_1 is labelled by a unitary diagram, d_i, then the formula for D_1 contains the sub-formula $\mathit{form}(d_3) \wedge (\mathit{form}(d_5) \vee \mathit{form}(d_6))$. By the distributivity of $\wedge$ over $\vee$, this is equivalent to $(\mathit{form}(d_3) \wedge \mathit{form}(d_5)) \vee (\mathit{form}(d_3) \wedge \mathit{form}(d_6))$, which is a sub-formula of the formula for D_2. Thus, the diagrams D_1 and D_2 have different structure but equivalent meanings. In this section we define an inference rule that transforms D_1 to D_2.

We will analyse the series of operations that are required to transform D_1 to D_2 in Figure 4.63. First, we identify an outer nested non-linear $\vee$-labelled connective; in our example the only candidate is n_4. Next, we remove the branch of the nearest inner non-linear $\vee$-labelled node, n_2, that contains the outer node n_4. For our example, the result of removing this sub-diagram from D_1 is shown in Figure 4.64, diagram D_1'. Next, we take the sub-diagram that we have just removed from D_1 and remove n_4 and its descendants, calling the result H (for

Head, since this sub-diagram is the head of the node to be removed); see Figure 4.64.

The next step is to take the set of sub-diagrams induced by immediate descendants of n_4 in the original diagram, D_1, (see the two diagrams labelled $\mathcal{T}$, for *Tails*, in Figure 4.64) and attach each of them to a copy of H. In our example this results in the two diagrams labelled $\mathcal{HT}$, shown in Figure 4.64. Finally, this set of diagrams is attached to n_2 in D_1', resulting in diagram D_2, Figure 4.63.

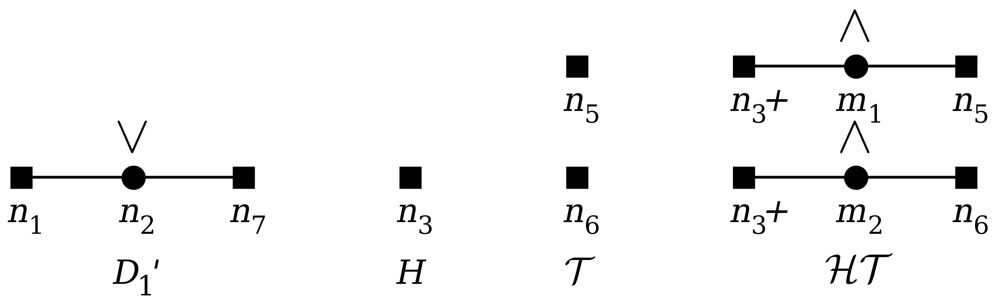


Figure 4.64: Intermediate steps in removing a nested non-linear $\vee$-labelled connective.

We define the rule which removes $\vee$-labelled nodes so that it removes outer nodes with respect to the nearest inner non-linear node. The diagram in Figure 4.65 contains three non-linear $\vee$-labelled connectives, n_2, n_4 and n_6. The node n_4 is nested within n_2, and n_6 is nested within both n_2 and n_4. There is a path from n_4 to n_6 that contains no non-linear connectives. Furthermore, each branch of n_6 is a linear sub-diagram of D_1. Syntactically, these are the conditions required for n_6 to be an outer nested non-linear connective and for n_4 to be the nearest inner non-linear node to n_6. Thus, when removing $\vee$-labelled connectives from the diagram in Figure 4.65, we begin by removing n_6 with respect to n_4.

Rule 15: Remove disjunct. The following inference rule removes a single non-linear $\vee$-labelled node. We can apply the rule repeatedly to a diagram in $\wedge$-linear normal form to obtain a diagram that contains at most one non-linear $\vee$-labelled node.

Formal description. Let D_1 be a generalized diagram in $\wedge$-linear normal form, let $H \subseteq D_1$ and let n_1 and n_2 be nodes in D_1 such that the following is true:

1. n_1 and n_2 are non-linear $\vee$-labelled connectives of D_1 such that $n_1 \in$

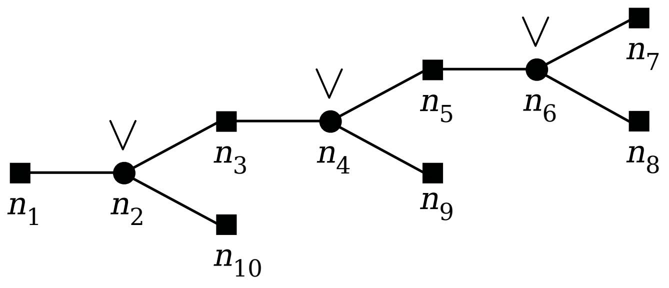


Figure 4.65: Multiple nested non-linear $\vee$-labelled connectives.

$Anc(n_2, D_1)$,

2. H is the sub-diagram of D_1 whose nodes are in $Des(n_1, D_1) \cap Anc(n_2, D_1)$,
3. each connective-labelled node in H is linear in D_1, and
4. each member of the set of sub-diagrams of D_1 induced by the immediate descendants of n_2 is a linear sub-diagram of D_1.

Let r be the root node of H, let D_r be the sub-diagram of D_1 induced by r, and let D_1' be the diagram obtained by using transformation 13, remove sub-diagram, to remove D_r from D_1: $D_1 \xrightarrow{-D_r} D_1'$. Let $\mathcal{T}$ be the set of sub-diagrams of D_1 induced by the immediate descendants of n_2, and let $\mathcal{HT}$ be the result attaching each element of $\mathcal{T}$ to H using $\wedge$:

$$\mathcal{HT} = \{H \xrightarrow{+(n_l,\wedge,T_i)} HT_i : T_i \in \mathcal{T}\},$$

where n_l is the (unique) leaf node of H. Let D_2 be the diagram obtained by attaching $\mathcal{HT}$ to n_1 in D_1' using transformation 19, attach diagrams to connective: $D_1' \xrightarrow{+(n_1,\mathcal{HT})} D_2$. Then D_1 can be replaced by D_2 and vice versa.

The next rule relabels linear $\vee$-labelled nodes by $\wedge$. This is justified by the fact that, whenever an $\vee$-labelled connective has exactly one branch, its interpretation is equivalent to that of a node labelled by $\wedge$. When interpreting a diagram, the information provided by the set of sub-diagrams which are descendants of an $\vee$-labelled node is interpreted in conjunction with that which came before it.

In Figure 4.66, if diagram-labelled nodes in the diagram D_1, n_i, are labelled by unitary diagrams, d_i, the formula for D_1 is given by $form(d_1) \wedge (form(d_3) \vee$

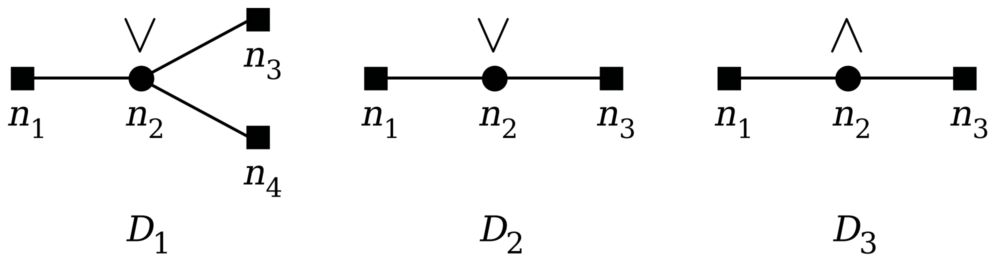


Figure 4.66: Relabelling a linear ∨-labelled node.

$\mathit{form}(d_4))$, or

$$\mathit{form}(d_1) \wedge (\bigvee_{d \in \{d_3, d_4\}} d).$$

Removing one of the sub-diagrams attached to the ∨-labelled node, n_2, results in diagram D_2, Figure 4.66. The formula for D_2 is given by

$$\mathit{form}(d_1) \wedge (\bigvee_{d \in \{d_3\}} d).$$

The ∨ connective is redundant in this formula and the formula for D_2 is equivalent to $\mathit{form}(d_1) \wedge \mathit{form}(d_3)$. This is the same as the formula for D_3, which is a copy of D_2 in which n_2 is relabelled by ∧.

Rule 16: Relabel linear disjunct. We now define the rule which allows us to relabel linear ∨-labelled nodes.
Formal description. Let D_1 be a generalized diagram that contains a linear ∨-labelled node, n. Let D_2 be the diagram obtained by using transformation 18, relabel node, to relabel n in D_1 by ∧: $D_1 \xrightarrow{l(n,\wedge)} D_2$. Then D_1 can be replaced by D_2 and vice versa.

4.4.2 The trivial prefix

In the previous section we defined rules that can be used to ensure that a diagram, D, contains at most one ∨-labelled node. We have defined disjunctive normal form so that, if D is a linear diagram, then D is in disjunctive normal form. If not, having reduced the number of ∨-labelled nodes to one, it remains for us to supply D with a trivial prefix. That is, to manipulate D so that a single node, labelled by the trivial diagram, appears before the single disjunctive-labelled node. We have already defined a rule which provides D with a trivial root: rule 5, attach to trivial diagram. After using this rule, we take the unitary diagram, d, immediately before the unique non-linear $\vee$-labelled node in D, n, and push copies of d past n until the original node labelled by d can be removed without changing the meaning of D. We repeat this process until the root node, labelled by the trivial diagram, is the only node before n.

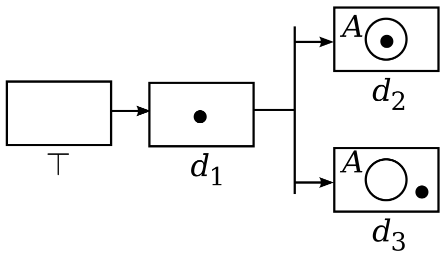


Figure 4.67: A generalized diagram with a root node labelled by the trivial diagram.

The diagram shown in 4.67 is in $\wedge$-linear and pushed syntax normal forms, contains a single non-linear $\vee$-labelled node, and has a root node labelled by the trivial diagram. In order to transform this diagram to disjunctive normal form, we push copies of the unitary diagram, d_1, which is the immediate ancestor of the $\vee$-labelled node, past that node into every disjunctive branch. This results in the diagram shown in Figure 4.68.

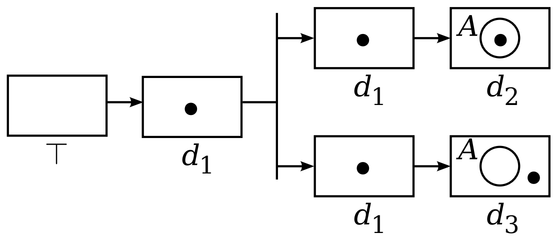


Figure 4.68: Pushing copies of a unitary diagram past the $\vee$-labelled node.

The formula for the diagram in Figure 4.68 is as follows:

$$\top \wedge \mathit{form}(d_1) \wedge ((\mathit{form}(d_1) \wedge \mathit{form}(d_2)) \vee (\mathit{form}(d_1) \wedge \mathit{form}(d_3))).$$

By the distributivity of $\wedge$ over $\vee$, the first occurrence of $form(d_1)$ can be removed from that formula without changing its meaning. Thus, we can remove the immediate ancestor of the $\vee$-labelled node without changing the meaning of the diagram, resulting in the diagram shown in Figure 4.69, which is in disjunctive normal form.

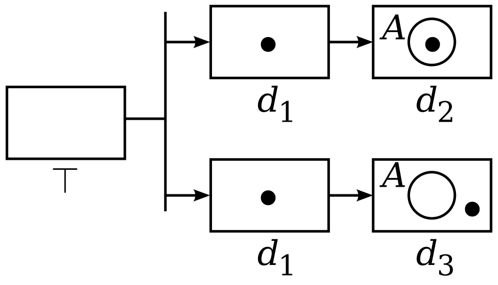


Figure 4.69: A generalized diagram in disjunctive normal form.

Rule 17: Push node. First, we define the rule that allows us to push copies of a unitary diagram past a connective-labelled node.
Formal description. Let $D_1 = (V_1, W_1, E_1, l_1)$ be a generalized diagram that contains nodes n_1, $n_\diamond$ and n_2 such that the following is true:

1. n_1 is the immediate ancestor of $n_\diamond$,
2. $n_\diamond$ is the immediate ancestor of n_2,
3. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2, respectively.

Let $D_1' = (V_1', W_1', E_1', l_1')$ be the sub-diagram of D_1 induced by n_2. Let m_1 and m_2 be nodes not in D_1, and let $D_1'' = (V_2, W_2, E_2, l_2)$ be the generalized diagram which satisfies the following:

1. $V_2 = \{m_1\} \cup V_1'$,
2. $W_2 = \{m_2\} \cup W_1'$,
3. $E_2 = \{(m_1, m_2), (m_2, root(D_1'))\} \cup E_1'$, and
4. $l_2 = \{(m_1, d_1), (m_2, \wedge)\} \cup l_1'$.

Let D_2 be the diagram obtained by using transformation 22, replace sub-diagram, to replace D_1' by X_2 in D_1'': $D_1 \xrightarrow{\rho(D_1', D_1'')} D_2$. Then D_1 can be replaced by D_2 and vice versa.

Rule 18: Discard redundant node. After using the previous rule to push copies of a node past a non-linear $\vee$-labelled node into every branch, the following rule allows us to remove the original node.

Formal description. Let D_1 be a generalized constraint diagram containing nodes n_1 and $n_\vee$ such that the following is true.

1. n_1 is not the root node of D_1,
2. n_1 is labelled by the unitary diagram, d_1,
3. n_1 is the immediate ancestor of $n_\vee$, which is a non-linear $\vee$-labelled node, and
4. each node, n_i, in the set of immediate diagram-labelled descendants of n_1 is labelled by d_1.

Let D_2 be the diagram obtained by removing n_1 from D_1 using transformation 15, remove diagram-labelled node: $D_1 \xrightarrow{-n_1} D_2$. Then D_1 can be replaced by D_2 and vice versa.

4.4.3 Validity of the inference rules required by disjunctive normal form

We now show that the rules which produce disjunctive normal form are sound.

Theorem 4.4.1: Rule 15, remove disjunct, is valid. Let D_1 be a generalized diagram in $\wedge$-linear normal form, let $H \subseteq D_1$ and let n_1 and n_2 be nodes in D_1 such that the following is true:

1. n_1 and n_2 are non-linear $\vee$-labelled connectives of D_1 such that $n_1 \in Anc(n_2, D_1)$,
2. H is the sub-diagram of D_1 whose nodes are in $Des(n_1, D_1) \cap Anc(n_2, D_1)$,
3. H is a linear sub-diagram of D_1, and

4. each member of the set of the sub-diagrams of D_1 induced by the immediate descendants of n_2 is a linear sub-diagram of D_1.

Let D_2 be the diagram obtained by using the rule to remove n_2 from D_1. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We will begin by showing that if I' is valid for D_1 then it is also valid for D_2. As in the definition of the rule, let $\mathcal{T}$ be the set of the sub-diagrams of D_1 induced by the immediate descendants of n_2, and let $\mathcal{HT}$ be the result of attaching each element of $\mathcal{T}$ to H using $\wedge$:

$$\mathcal{HT} = \{H \xrightarrow{+(n_l,\wedge,T_i)} HT_i : T_i \in \mathcal{T}\},$$

where n_l is the (unique) leaf node of H. We know that H is a linear sub-diagram of D_1. We also know that each element of $\mathcal{T}$ is a linear sub-diagram of D_1. Let D_{1X} be the sub-diagram of D_1 induced by n_1 and let d_1 be the unitary diagram labelling n_1. Then D_{1X} has the following formula:

$$\mathit{form}(d_1) \wedge \mathit{form}(H) \wedge (\bigvee_{T_i \in \mathcal{T}} \mathit{form}(T_i)). \tag{4.22}$$

Each diagram in the set $\mathcal{HT}$ is the result of attaching an element of $\mathcal{T}$ to the leaf node of H. The elements of $\mathcal{T}$ and H are all linear diagrams. Thus, the formula for the disjunction of the diagrams in $\mathcal{HT}$ is as follows:

$$\bigvee_{T_i \in \mathcal{T}} (\mathit{form}(H) \wedge \mathit{form}(T_i)). \tag{4.23}$$

In diagram D_2, the set of diagrams in $\mathcal{HT}$ is attached to the node n_1. Therefore, D_2 contains the following sub-formula:

$$\mathit{form}(d_1) \wedge \bigvee_{T_i \in \mathcal{T}} (\mathit{form}(H) \wedge \mathit{form}(T_i)). \tag{4.24}$$

By the distributivity of $\wedge$ over $\vee$, (4.22) is equivalent to (4.24). Diagram D_2 is a copy of D_1 in which the descendants of n_1 are replaced by the disjunction of the diagrams in the set $\mathcal{HT}$. In terms of the formulae, that is to say that (4.22) is replaced by (4.24). It follows that the formula for D_2 is true under I' whenever the formula for D_1 is true under I', and so $D_1 \vDash D_2$. We can show that $D_2 \vDash D_1$ by a similar argument and thus $D_1 \equiv_{\vDash} D_2$ as required. □

Theorem 4.4.2: Rule 16, relabel linear disjunct, is valid. Let D_1 be a generalized diagram that contains a node n which is a linear $\vee$-labelled connective. Let D_2 be the diagram obtained by using the rule to relabel n in D_1 by $\wedge$. Then $D_1 \equiv_{\vDash} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We begin by showing that the formula for D_2 is true under I' whenever it is true for D_1. Let n_A and n_D be the immediate ancestor and descendant of n, respectively. D_1 has the following sub-formula:

$$\mathit{form}(n_A) \wedge (\bigvee_{n' \in \{n_D\}} \mathit{form}(n')).$$

Since $\{n_D\}$ has a single element, this is equivalent to $\mathit{form}(n_A) \wedge \mathit{form}(n_D)$, which is the sub-formula for the corresponding sub-diagram of D_2. Thus the formula for D_2 is true under I' whenever the formula for D_1 is true under I', and so $D_1 \vDash D_2$. We can see that $D_2 \vDash D_1$ by a similar argument and thus $D_1 \equiv_{\vDash} D_2$. □

Theorem 4.4.3: Rule 17, push node, is valid. Let D_1 be a generalized diagram that contains nodes n_1, $n_\diamond$ and n_2 such that the following is true:

1. n_1 is the immediate ancestor of $n_\diamond$,
2. $n_\diamond$ is the immediate ancestor of n_2,
3. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2, respectively.

Let D_2 be the diagram obtained by using the rule to push a copy of n_1 past $n_\diamond$. Then D_1 can be replaced by D_2 and vice versa.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. Let d_1 and d_2 be the unitary diagrams labelling n_1 and n_2, respectively. We will show that the formula for D_2 is true under I' whenever the formula for D_1 is true by considering the cases for $\diamond$, the connective used to label $n_\diamond$. Assume that $n_\diamond$ is labelled by $\vee$ and is a non-linear node. Let $\mathcal{X}$ be the set of sub-diagrams of D_1

induced by the immediate descendants of $n_\diamond$. Then the formula for D_1 contains the following sub-formula:

$$\mathit{form}(d_1) \wedge \bigvee_{X_i \in \mathcal{X}} \mathit{form}(X_i). \tag{4.25}$$

As in the definition of the rule, let $D'_1 = (V'_1, W'_1, E'_1, l'_1)$ be the sub-diagram of D_1 induced by n_2. Furthermore, let m_1 and m_2 be nodes not in D_1, and let $D''_1 = (V_2, W_2, E_2, l_2)$ be the generalized diagram which satisfies the following:

1. $V_2 = \{m_1\} \cup V'_1$,
2. $W_2 = \{m_2\} \cup W'_1$,
3. $E_2 = \{(m_1, m_2), (m_2, \mathit{root}(D'_1))\} \cup E'_1$, and
4. $l_2 = \{(m_1, d_1), (m_2, \wedge)\} \cup l'_1$.

Since $D'_1 \in \mathcal{X}$, we can rewrite (4.25) as follows:

$$\mathit{form}(d_1) \wedge \big(\mathit{form}(D'_1) \vee \bigvee_{X_i \in \mathcal{X} - \{D'_1\}} \mathit{form}(X_i)\big). \tag{4.26}$$

The formula for D_2 is equivalent to a copy of the formula for D_1 in which the sub-formula for D'_1 is replaced by that for D''_1, which is equal to $\mathit{form}(d_1) \wedge \mathit{form}(D'_1)$. So, the formula for D_2 is a copy of that for D_1 in which (4.26) is replaced by the following.

$$\mathit{form}(d_1) \wedge \big((\mathit{form}(d_1) \wedge \mathit{form}(D'_1)) \vee \bigvee_{X_i \in \mathcal{X} - \{D'_1\}} \mathit{form}(D_i)\big). \tag{4.27}$$

Again, we can see that (4.27) is true whenever (4.26) is true, and vice versa. Therefore, $D_1 \equiv_{\models} D_2$ if $n_\diamond$ is a non-linear $\vee$-labelled node. If $n_\diamond$ is a non-linear $\wedge$-labelled node, the argument is similar.

Assume that $n_\diamond$ is a linear $\wedge$-labelled node. Then the formula for D_1 contains the following sub-formula:

$$\mathit{form}(d_1) \wedge \mathit{form}(d_2). \tag{4.28}$$

The formula for D_2 is a copy of that for D_1 in which (4.28) is replaced by

$$\mathit{form}(d_1) \wedge \mathit{form}(d_1) \wedge \mathit{form}(d_2). \tag{4.29}$$

Clearly, (4.29) is true whenever (4.28) is true, and so I' is valid for D_2 if $n_\diamond$ is a linear $\wedge$-labelled node. Equally, (4.28) is true whenever (4.29) is true, and so any valid extension for D_2 is valid for D_1, and $D_1 \equiv_\models D_2$ if $n_\diamond$ is a linear $\wedge$-labelled node. If $n_\diamond$ is a linear $\vee$-labelled node, the argument is similar. Thus, $D_1 \equiv_\models D_2$ as required. □

Theorem 4.4.4: Rule 18, discard redundant node, is valid. Let D_1 be a generalized constraint diagram containing nodes n_1 and $n_\vee$ such that the following is true.

1. n_1 is not the root node of D_1,
2. n_1 is labelled by the unitary diagram, d_1,
3. n_1 is the immediate ancestor of $n_\vee$, which is a non-linear $\vee$-labelled node, and
4. each node, n_i, in the set of immediate diagram-labelled descendants of n_1 is labelled by d_1.

Let D_2 be the diagram obtained by using the rule to remove n_1 from D_1. Then D_1 can be replaced by D_2 and vice versa.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We begin by showing that the formula for D_2 is true under I' whenever the formula for D_1 is true under I'. Let $\mathcal{X}$ be the set of sub-diagrams of D_1 induced by the immediate diagram-labelled descendants of n_1. Then the formula for D_1 contains the following sub-formula:

$$\mathit{form}(d_1) \wedge \Big(\bigvee_{X_i \in \mathcal{X}} \mathit{form}(X_i) \Big). \tag{4.30}$$

The root node of each member of $\mathcal{X}$ is labelled by d_1. Thus, for each sub-diagram, X_i, of D_1 in $\mathcal{X}$, X_i has the formula

$$\mathit{form}(d_1) \wedge Y_i, \tag{4.31}$$

where Y_i is the formula for the sub-diagram or sub-diagrams of X_i induced by the immediate diagram-labelled descendants of the root node of X_i. (If the immediate descendant of the root node of X_i, say x_2, is a non-linear connective, then Y_i is a disjunction or conjunction of clauses, depending on the label of x_2.) Rewriting (4.30) by (4.31),

$$\mathit{form}(d_1) \wedge \bigvee_{X_i \in \mathcal{X}} (\mathit{form}(d_1) \wedge Y_i). \tag{4.32}$$

By the distributivity of $\wedge$ over $\vee$, and by idempotency, this is equivalent to the following:

$$\bigvee_{X_i \in \mathcal{X}} (\mathit{form}(d_1) \wedge Y_i). \tag{4.33}$$

The formula for D_2 is equivalent to a copy of that of D_1 in which (4.32) is replaced by (4.33). As we have shown that these are equivalent sub-formulae, the formula for D_2 is true whenever the formula for D_1 is true, and so $D_1 \vDash D_2$. By a similar argument, any valid extension for D_2 is also valid for D_1, and so $D_2 \vDash D_1$ and $D_1 \equiv_{\vDash} D_2$ as required. $\square$

4.4.4 Obtaining disjunctive normal form

We can now use the definitions of the inference rules in sections 4.4.1 and 4.4.2 to state the algorithm which transforms an initial diagram, D_1, which is in $\wedge$-linear and pushed syntax normal forms, to a diagram, D_2, where D_2 is in disjunctive normal form. We then show that the algorithm ensures that D_2 has an equivalent meaning to D_1, and that the algorithm maintains the syntactic conditions of D_1 implied by $\wedge$-linear and pushed syntax normal forms.

The latter point means it must be true that D_2 contains no non-linear $\wedge$-labelled nodes, and that $d_i \subseteq_S d_j$ or $d_j = \bot$ for all unitary diagrams, d_i and d_j, where d_i labels a node in D_2 and d_j is an immediate diagram-labelled descendant of d_i. In fact, $\wedge$-linear normal form is implied by the definition of disjunctive normal form, and so the obligation is to show that the process of obtaining disjunctive normal form does not interfere with the property of pushed syntax normal form. We address this next, at the same time as showing that the rules we have are sufficient to remove $\vee$-labelled nodes.

Lemma 4.4.1. Let D_1 be a generalized diagram in $\wedge$-linear normal form and pushed syntax normal form. Then there is a sequence of inference rules which can be used to transform D_1 into D_2, where D_2 satisfies the following:

1. D_2 contains at most one $\vee$-labelled node,
2. $D_1 \equiv_{\models} D_2$, and
3. D_2 is in pushed syntax normal form.

Proof. To show (1), we know that D_1 is in $\wedge$-linear normal form. Thus, we can use rule 15, remove disjunct, to remove all outer nested non-linear $\vee$-labelled nodes from D_1 to produce a diagram, D_1', which contains at most one non-linear $\vee$-labelled node. We can then use rule 16 to relabel any linear $\vee$-labelled nodes by $\wedge$ in D_1', obtaining diagram D_2, containing at most one $\vee$-labelled node.

Next, we show that (2) is true for the diagram D_2 obtained in the previous step. The diagram D_1' is produced by repeated application of rule 15, remove disjunct. Let $\hat{D}_1$ be the diagram obtained by removing a single non-linear $\vee$-labelled node from D_1. By theorem 4.4.1, $D_1 \equiv_{\models} \hat{D}_1$. It follows that $D_1 \equiv_{\models} D_1'$. By theorem 4.4.2, we know that rule 16 is an equivalence and so, by a similar reasoning, $D_1' \equiv_{\models} D_2$. Thus, $D_1 \equiv_{\models} D_2$ as required.

Finally, it remains to show (3): that D_2 is in pushed syntax normal form. We know that D_1 is in this normal form, and we can reason that the only stage at which we could have affected this is by application of rule 15, remove disjunct. This rule works by identifying the 'head', a linear diagram, and 'tails', a set of linear diagrams, which are the prefix and suffix of a non-linear $\vee$-labelled node, respectively. The 'head' and 'tails' are then reconnected in a diagram which is the conclusion of the rule. Since the leaf node of the 'head' is the immediate diagram-labelled ancestor of the root node of each member of the 'tails', D_2 is in pushed syntax normal form. □

Next, we show that the use of rule 17, push node, does not affect the conditions of $\wedge$-linear and pushed syntax normal form. This is a necessary part of the argument that says that the transformation to disjunctive normal form maintains the normal form conditions of the original diagram.

Lemma 4.4.2. Let $D_1 = (V, W, E, l)$ be a generalized diagram in $\wedge$-linear normal form and pushed syntax normal form that contains nodes n_1, $n_\diamond$ and n_2 such that the following is true:

1. n_1 is the immediate ancestor of $n_\diamond$,
2. $n_\diamond$ is the immediate ancestor of n_2,
3. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2, respectively.

Let D_2 be the diagram obtained by using rule 17, push node, to push a copy of n_1 past $n_\diamond$. Then D_2 is in $\wedge$-linear normal and pushed syntax normal forms.

Proof. The push node rule does not introduce any new non-linear $\wedge$-labelled nodes and so, trivially, D_2 is in $\wedge$-linear normal form. Let $\mathcal{M}$ be the set of immediate diagram-labelled descendants of n_1 in D_1. Since D_1 is in pushed syntax normal form then, for each node, $m_i \in \mathcal{M}$, either $l(n_1) \subseteq_S l(m_i)$ or $m_i = \bot$. Let $\mathcal{M}'$ be the set of immediate diagram-labelled descendants of n_1 in D_2. $\mathcal{M}'$ does not contain n_2, but contains a node labelled $l(n_1)$. By the definition of $\subseteq_S$, the syntactic sub-diagram relation is reflexive, so $l(n_1) \subseteq_S l(n_1)$. The other diagram-labelled nodes in D_2 are not affected, and so D_2 is in pushed syntax normal form. □

We are now able to state the algorithm used to transform a generalized diagram in $\wedge$-linear normal form and pushed syntax normal form, D_1, to a second diagram, D_2, where D_2 is in disjunctive normal form.

Algorithm 3 (Pushed syntax normal form to disjunctive normal form)**.** Let D_1 be a generalized diagram in $\wedge$-linear normal form and pushed syntax normal form. Transform D_1 to a second diagram, D_2, using the following steps:

1. If D_1 is a linear diagram, let $D_2 = D_1$ and we are done.
2. Otherwise, D_1 is a non-linear diagram. Use the steps described in lemma 4.4.1 to transform D_1 into a diagram, E_1, where E_1 is in $\wedge$-linear normal form and pushed syntax normal form.
3. Next, use rule 5, attach to trivial diagram, to prefix E_1 with a node labelled by $\top$, giving diagram E_2.
4. Let $n_\vee$ be the unique non-linear $\vee$-labelled node in E_2, let n_1 be the immediate ancestor of $n_\vee$ and let d_1 be the unitary diagram labelling n_1. If n_1 is the root node of E_2, let $D_2 = E_2$ and we are done.

5. Otherwise, n_1 is not the root node of E_2. If each n_i in the immediate diagram-labelled descendants of n_1 is labelled by a unitary diagram d_i, such that $d_i = d_1$, use rule 18 to remove n_1 from E_2, giving E_3. Continue at (4) in the context of E_3.

6. Otherwise, n_1 is not the root node of E_2 and there exists a diagram-labelled descendant, n_2, of n_1 labelled by a unitary diagram, say d_2, where $d_1 \neq d_2$. Then we use rule 17, push node, to push a copy of d_1 past $n_\vee$ towards n_2, giving diagram E_3. Continue at (5) in the context of E_3.

Next we show that we can use the above algorithm to transform a generalized diagram in $\wedge$-linear normal form and pushed syntax normal form, D_1, into a second diagram, D_2, where D_2 is in $\wedge$-linear, pushed syntax and disjunctive normal forms, and has an equivalent meaning to D_1. Also, we show that algorithm 3 terminates.

Theorem 4.4.5: Let D_1 be a generalized diagram in $\wedge$-linear normal form and pushed syntax normal form. Let D_2 be the diagram obtained by using algorithm 3 to transform D_1. Then the following is true:

1. D_2 is in $\wedge$-linear normal form,

2. D_2 is in pushed syntax normal form,

3. D_2 is in disjunctive normal form,

4. $D_1 \equiv_\models D_2$, and

5. algorithm 3 terminates.

Proof. 1. If the algorithm terminates at step (1), then the four conditions are trivially true.

2. In step (2), D_1 is transformed into a diagram, E_1, where E_1 contains a single non-linear disjunct and is in $\wedge$-linear normal form and pushed syntax normal forms. By lemma 4.4.1, $D_1 \equiv_\models E_1$. Clearly, the normal forms are not affected by this operation, and E_1 is in $\wedge$-linear and pushed syntax normal form.

3. In step (3) we use rule 5, attach to trivial diagram, to prefix E_1 with a node labelled by $\top$, giving E_2. By theorem 4.3.1, $E_1 \equiv_{\models} E_2$. Again, the normal forms are not affected by this operation, and E_2 is in $\wedge$-linear and pushed syntax normal form.

4. Let $n_\vee$ be the unique non-linear $\vee$-labelled node in E_2, let n_1 be the immediate ancestor of $n_\vee$ and let d_1 be the unitary diagram labelling n_1. If n_1 is the root node of E_2, then E_2 is in $\wedge$-linear, pushed syntax and disjunctive normal forms. The process assigns $D_2 = E_2$ and terminates.

5. If we reach step (5) then n_1 is not the root node of E_2. If each n_i in the immediate diagram-labelled descendants of n_1 is labelled by a unitary diagram d_i, such that $d_i = d_1$, we use rule 18 to remove n_1 from E_2, giving E_3. By theorem 4.4.4, $E_2 \equiv_{\models} E_3$. The use of this rule does not affect the normal forms, and so E_3 is in $\wedge$-linear normal form and pushed syntax normal form. The process continues at (4) in the context of E_3.

6. Otherwise, n_1 is not the root node of E_2 and there exists a diagram-labelled descendant, n_2, of n_1, labelled by a unitary diagram, say d_2, where $d_1 \neq d_2$. Then we use rule 17, push node, to push a copy of d_1 past $n_\vee$ towards n_2, giving diagram E_3. By theorem 4.4.3, $E_2 \equiv_{\models} E_3$. By lemma 4.4.2, E_3 is in $\wedge$-linear normal form and pushed syntax normal form. The process continues at (5) in the context of E_3.

Since all of the inference rules employed are equivalences, $D_1 \equiv_{\models} D_2$. The inference rules do not affect the normal forms, and so D_2 is in $\wedge$-linear, pushed syntax and disjunctive normal form as required.

Next, we will show that the above algorithm describes a terminating process. If the process does not terminate at step (1), it proceeds to step (4). In step (4), we know that $n_\vee$ exists, since E_2 is a non-linear diagram in pushed syntax normal form. We also know that $n_\vee$ has a finite number of ancestors, since E_2 is finite. During the process, two conditions are tested:

a P: $n_\vee$ has exactly one ancestor, and

b Q: each node, n_i, in the immediate diagram-labelled descendants of n_1, is labelled by d_1.

The process terminates when $n_\vee$ is found to have a single ancestor. That is, in step (4), if P is found to be true, the process terminates. If the process reaches step (5), we know that P is not true, i.e., the number of ancestors of $n_\vee$ is greater than one. Q is true in step (5), however. The inference rules used mean that the number of ancestors of $n_\vee$ decreases by one, and the process continues at step (4). In step (6), both P and Q are false and there are, say, j immediate diagram-labelled descendants of n_1 which are not labelled by d_1, for some $j > 0$. The use of rule 17, push node, means that j reduces by one, and the process continues at step (5), where Q is tested again. So we can see that, at each step, the number of ancestors of $n_\vee$ is reduced or, if we are unable to take that step, the number of diagram-labelled descendants of n_1 which stop us from taking the former step is reduced. Thus, we have shown the following as required:

1. D_2 is in $\wedge$-linear normal form,
2. D_2 is in pushed syntax normal form,
3. D_2 is in disjunctive normal form,
4. $D_1 \equiv_\models D_2$, and
5. algorithm 3 terminates.

□

4.5 Reducing linear generalized diagrams

In this section we present the final stage of the decision procedure for the full existential fragment. This depends on an algorithm which, given a linear generalized diagram, D, in pushed syntax normal form reduces D to a generalized diagram, D', where D' has a single node and is equivalent in meaning to D. Clearly, D' is satisfiable if and only if the unitary diagram labelling its root node is satisfiable, and it also follows that D is satisfiable if and only if D' is satisfiable. Figure 4.70 shows the algorithm we will develop in this section in the context of the decision procedure.

Although the algorithm we will state deals solely with linear diagrams, we will apply it repeatedly to reduce the branches of a diagram in disjunctive normal

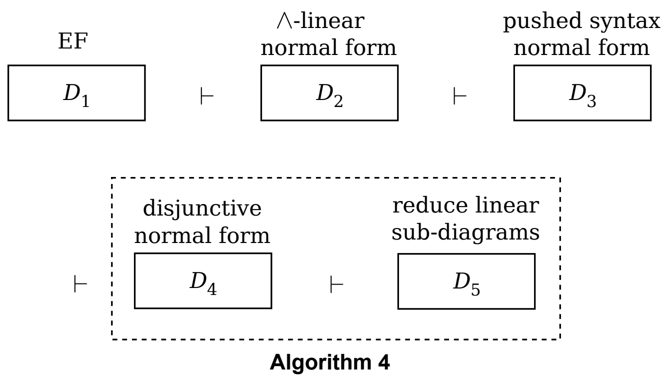
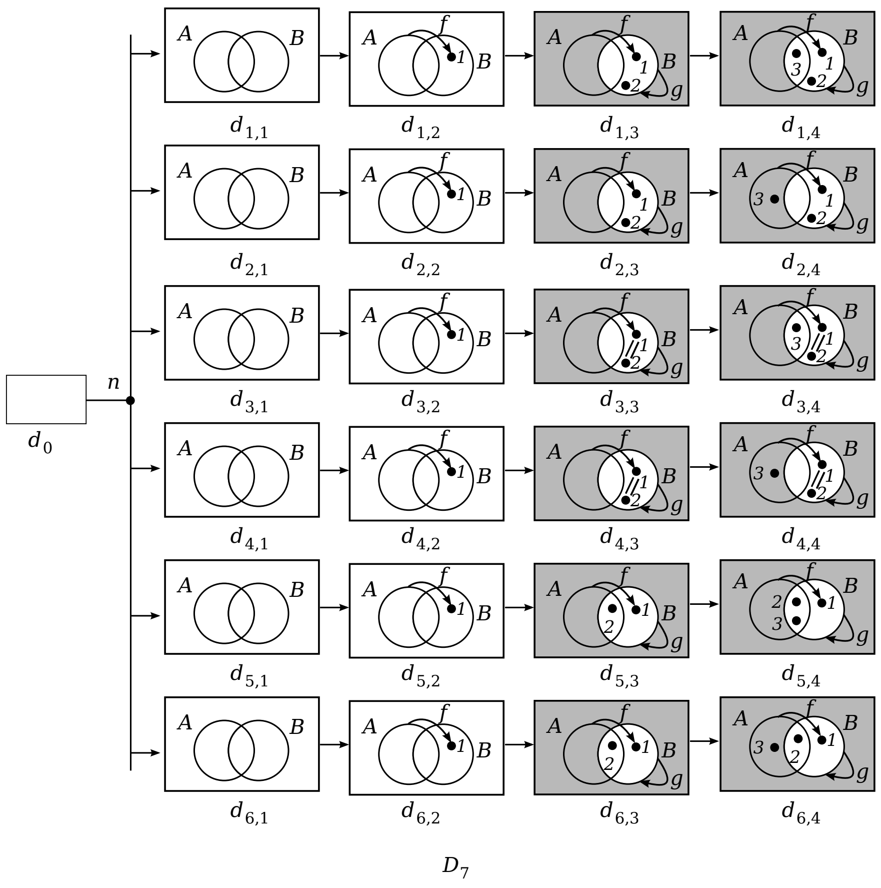


Figure 4.70: The algorithm which reduces linear generalized diagrams, in context.

form. From this, we are able to show that a generalized diagram in disjunctive normal form, D, is satisfiable if and only one of its branches is satisfiable. In Figure 4.71, diagram D_7 is the disjunctive normal form of our running example. Each branch of the disjunctive-labelled node, n, is a linear sub-diagram of D_7, and is also in pushed syntax normal form.

The branches of the $\vee$-labelled node, n, in diagram D_7 are in pushed syntax normal form. Thus, for each such branch, D', the unitary diagram, d_l, labelling the leaf node of D' contains every syntactic element found in its ancestors, or $d_l = \perp$. This means that, if D' is consistent, we can remove non-leaf diagram-labelled nodes from D', starting with the root node, without changing its meaning. Consider the unitary diagrams $d_{2,3}$ and $d_{2,4}$ in diagram D_7, Figure 4.71. In $d_{2,3}$, the zone $(\{A\}, \{B\})$ is shaded and untouched, whilst in $d_{2,4}$, the same zone is shaded and contains a spider. Clearly, these diagrams are inconsistent with respect to each other, although they are both satisfiable in their own right. We are unable to remove the node labelled by $d_{2,3}$ from this branch without changing the meaning of D_7 because to do so would be to discard the information that this branch is inconsistent. The same type of inconsistency exists between the pairs of unitary diagrams $d_{4,3}$ and $d_{4,4}$, and $d_{6,3}$ and $d_{6,4}$. In section 4.3.2, we encountered inconsistency between pairs of diagrams when pushing spiders into shaded zones.

As before, we record the inconsistencies by relabelling diagrams with $\perp$. In cases where no inconsistency is found, we remove diagram-labelled nodes until we reduce the linear sub-diagram to a single diagram-labelled node. Note that the step of removing nodes is not strictly necessary, and that we could achieve



d_0
n
$d_{1,1}$ $d_{1,2}$ $d_{1,3}$ $d_{1,4}$
$d_{2,1}$ $d_{2,2}$ $d_{2,3}$ $d_{2,4}$
$d_{3,1}$ $d_{3,2}$ $d_{3,3}$ $d_{3,4}$
$d_{4,1}$ $d_{4,2}$ $d_{4,3}$ $d_{4,4}$
$d_{5,1}$ $d_{5,2}$ $d_{5,3}$ $d_{5,4}$
$d_{6,1}$ $d_{6,2}$ $d_{6,3}$ $d_{6,4}$
D_7
A
B
f
g
1
2
3

Figure 4.71: A running example: linear sub-diagrams in pushed syntax normal form.

the same purpose by comparing pairs of unitary diagrams, looking for the kind of inconsistency seen in the previous example. However, removing nodes repeatedly means that the process always compares the root node of a linear sub-diagram to its immediate diagram-labelled descendant, which makes the process simple to describe.

We begin by defining the type of inconsistency that can exist between pairs of (potentially satisfiable) unitary diagrams. We then show that we can use the presence or absence of this property to reduce a linear diagram in pushed syntax normal form to a single node.

4.5.1 Removing syntax from a linear diagram

First, we develop the *sub-diagram* relation for pairs of unitary diagrams. The syntactic sub-diagram relation is given in definition 4.3.6, page 163: given two unitary diagrams, d_1 and d_2, d_1 is a syntactic sub-diagram of d_2 if and only if all of the syntax of d_1 appears within d_2. This does not imply that d_1 and d_2 are consistent with each other. We provide a syntactic characterisation of such consistency which, in conjunction with the notion of syntactic sub-diagrams, is the basis of the sub-diagram relation. Figure 4.72 shows two diagrams, d_1 and d_2, where $d_1 \subseteq_S d_2$. In this case, it is also true that the meanings of d_1 and d_2 are consistent, so we say that d_1 is a sub-diagram of d_2, denoted $d_1 \subseteq d_2$.

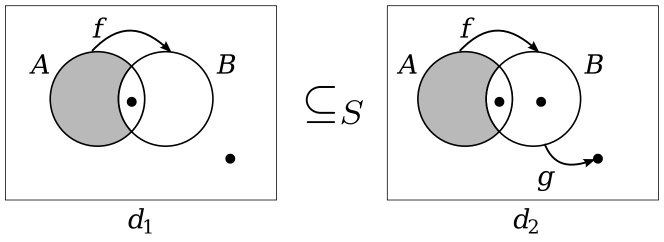


Figure 4.72: The sub-diagram relationship.

The definition of $\subseteq_S$ ensures that, given two unitary diagrams d_1 and d_2, if $d_1 \subseteq_S d_2$, then if a zone z is shaded in d_2 but not in d_1, there are at most as many spiders inhabiting z in d_1 as there are in d_2. However, if z is shaded in both d_1 and d_2, the definition of syntactic sub-diagrams does not ensure that there are the same number of spiders inhabiting z in each diagram. If this is not true then no interpretation can satisfy d_1 and d_2 at the same time. In Figure 4.73, $d_1 \subseteq_S d_2$ but, because of the different number of spiders inhabiting the shaded zone $(\{A\}, \{B\})$ in each diagram, the shaded zones condition cannot be true for both diagrams simultaneously in any interpretation.

The second syntactic condition that can cause two unitary diagrams, d_1 and d_2, to be inconsistent, despite the fact that $d_1 \subseteq_S d_2$, is that of ties between spiders. In Figure 4.74, $d_1 \subseteq_S d_2$, but while the spiders x and y represent distinct elements in d_1, they represent the same element in d_2. In this case, no interpretation can satisfy the spiders' distinctness condition for the two diagrams at the same time.

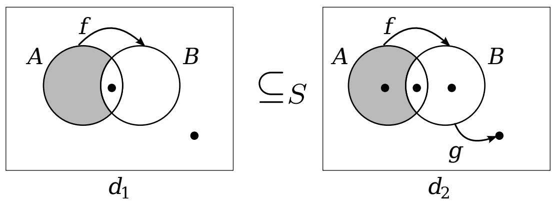


Figure 4.73: A syntactic sub-diagram which is inconsistent due to the shaded zones condition.

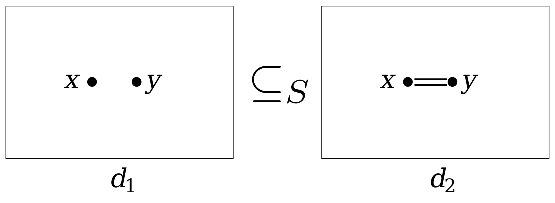


Figure 4.74: A syntactic sub-diagram which is inconsistent due to the spiders' distinctness condition.

Finally, we may have two unitary diagrams, d_1 and d_2, where $d_1 \subseteq_S d_2$ but where the habitat functions, η_{d_1} and η_{d_2}, are in disagreement. In Figure 4.75, diagram d_1, the spider labelled x inhabits the zone outside of the derived contour dc, but this is not true in diagram d_2. We can see that no interpretation can satisfy the spiders habitat condition for d_1 and d_2 simultaneously.

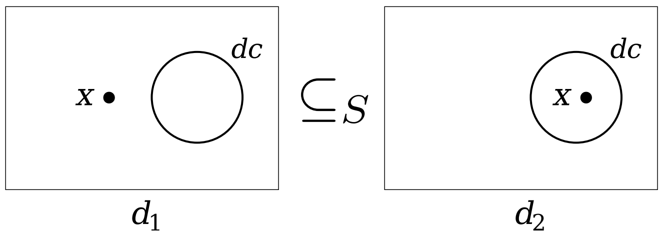


Figure 4.75: A syntactic sub-diagram which is inconsistent due to the spiders habitat condition.

The constraints we can infer from these examples enable us to give a syntactic characterisation of what it means for a pair of unitary diagrams, d_1 and d_2, where $d_1 \subseteq_S d_2$, to be consistent. We call this the *sub-diagram* relation.

Definition 4.5.1. Let d_1 and d_2 be generalized unitary diagrams such that the following is true:

1. $d_1 \subseteq_S d_2$,

2. for each $x \in S(d_1)$, $\eta_{d_1}(x) = \eta_{d_2}(x)$,

3. for each zone, z, which is shaded in d_1, any spider, x, which is in z in d_2 but not in d_1 is joined to a spider, y, by a tie in d_2, and y is present in d_1:

$$\forall z \in Z^*(d_2)\, x \in S(z, d_2) - S(z, d_1) \Rightarrow \exists y \in S(z, d_1) \cap [x]_{d_2},$$

   and

4. for each pair of spiders $x, y \in S(d_1)$, x and y are joined by a tie in d_1 if and only if they are joined by a tie in d_2:

$$\forall x, y \in S(d_1)\, ((x, y) \in \tau_{d_1} \Leftrightarrow (x, y) \in \tau_{d_2}).$$

Then we say that d_1 is a **sub-diagram** of d_2, denoted $d_1 \subseteq d_2$.

Next, we define an inference rule which allows us to relabel a diagram-labelled node, n_1, by $\top$ if n_1 has a unique diagram-labelled descendant, n_2, and the unitary diagram labelling n_1 is a sub-diagram of the unitary diagram labelling n_2.

Rule 19: Relabel sub-diagram. In Figure 4.76, diagram D_1 is composed of two nodes labelled by the unitary diagrams d_1 and d_2. Diagram d_1 is a sub-diagram of d_2. This means, informally, that the information provided by d_1 is a subset of that provided by d_2. In particular, $d_2 \vDash d_1$, so $\mathit{form}(d_2) \wedge \mathit{form}(d_1)$ is equivalent to $\mathit{form}(d_2)$. Informally, we can throw away d_1 without weakening information, and this justifies a rule that will produce D_2 from D_1, where D_2 is a copy of D_1 in which the node labelled by d_1 is relabelled by the trivial diagram.
Formal description. Let D_1 be a linear generalized diagram that contains nodes n_1 and n_2 such that the following is true:

1. n_1 is the immediate diagram-labelled ancestor of n_2,

2. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2 respectively, and

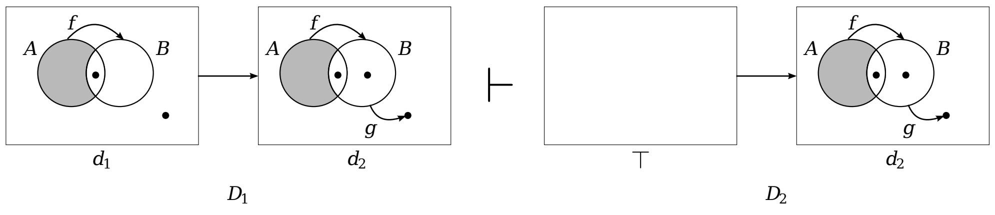


Figure 4.76: An application of the *relabel sub-diagram* rule.

3. $d_1 \subseteq d_2$.

Let D_2 be the diagram obtained by using transformation 18, relabel node, to label n_1 in D_1 by $\top$: $D_1 \xrightarrow{l(n_1,\top)} D_2$. Then D_1 can be replaced by D_2 and vice versa.

The counterpart to the above rule is the fact that, given two nodes, n_1 and n_2, within a linear diagram, D, where n_1 and n_2 are labelled by unitary diagrams d_1 and d_2 respectively, if $d_1 \subseteq_S d_2$ but $d_1 \not\subseteq d_2$, then D is unsatisfiable. We use this fact to define a rule which can be used to simultaneously remove n_2 and its descendants from D, and to relabel n_1 by $\perp$. Note that this implication holds in one direction only; it may be that, for each unitary diagram d_j and immediate diagram-labelled ancestor, d_i, of d_j in D, $d_i \subseteq d_j$, but that D is unsatisfiable. The sub-diagram relation provides information about the consistency of two diagrams *vis á vis* each other, but not about the satisfiability of the unitary diagrams involved.

Rule 20: Relabel inconsistent sub-diagram. In Figure 4.77, diagram D_1 contains nodes labelled by the unitary diagrams d_1 and d_2. Diagram D_1 is a linear diagram and $d_1 \subseteq_S d_2$, so D_1 is in pushed syntax normal form. In diagram d_1, the spiders labelled x and y are not joined by tie, whilst in diagram d_1, the same spiders are joined by a tie. Thus, $d_1 \not\subseteq d_2$ and d_1 and d_2 are inconsistent with each other. We will go on to show (in lemma 4.5.2) that D_1 is unsatisfiable and that we are justified in replacing D_1 by $\perp$.

Formal description. Let D_1 be a linear generalized diagram in pushed syntax normal form, containing nodes n_1 and n_2 which satisfy the following:

1. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2, respectively,

2. n_1 is the immediate diagram-labelled ancestor of n_2, and

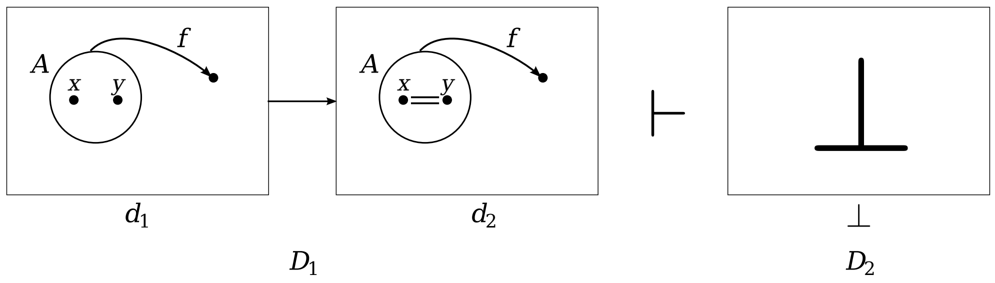


Figure 4.77: An application of the *relabel inconsistent sub-diagram* rule.

3. $d_1 \not\subseteq d_2$.

Let D_2 be the unitary diagram which consists of a single node labelled by $\perp$. Then D_1 can be replaced by D_2.

The final rule we require when reducing linear diagrams is one which removes the root node, n, of a linear diagram when n is labelled by $\top$.

Rule 21: Remove root node. If a node, n, of a linear generalized diagram, D, is labelled by the trivial diagram, we can remove n without changing the meaning of D. This follows immediately from the interpretation of $\top$, which is trivially true. Since we will use this rule in the algorithmic process to remove the root node of a linear diagram, we define the rule in this context.

Formal description. Let D_1 be a linear generalized diagram which contains nodes n_1 and n_2 such that the following is true:

1. n_1 is the root node of D_1 and is labelled by $\top$,

2. n_2 is the immediate diagram-labelled descendant of n_1.

Let D_2 be the diagram obtained by using transformation 14, remove root node, to remove n_1 from D_1: $D_1 \xrightarrow{-n_1} D_2$. Then D_1 can be replaced by D_2 and vice versa.

4.5.2 Validity of the inference rules required to reduce linear diagrams

We will now show that the rules which relabel sub-diagrams and remove nodes labelled by the trivial diagram are sound. First, we establish two results concerning the sub-diagram relation and satisfiability. The first of these shows that, given unitary diagrams d_1 and d_2, if d_1 is a sub-diagram of d_2, then the semantic formula for d_2 implies that of d_1.

Lemma 4.5.1. Let D be a generalized diagram that contains nodes n_1 and n_2, labelled by unitary diagrams d_1 and d_2 respectively, such that $d_1 \subseteq d_2$. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. Then, under I', $\mathit{form}(d_2) \Rightarrow \mathit{form}(d_1)$.

Proof. Assume $\mathit{form}(d_2)$ is true under I'. The diagrams d_1 and d_2 have the same zone set, and so the plane tiling condition is true for d_1 since it is true for d_2. By the definition of sub-diagrams we know that, for all spiders $x \in S(d_1)$, $\eta_{d_1}(x) = \eta_{d_2}(x)$. Thus, the spiders' habitat condition is true in d_1 since it is true in d_2. By the same definition, we also know that two spiders in $S(d_1)$, x and y, are joined by a tie if and only if they are joined by a tie in d_2. Let x and y be spiders in $S(d_1)$. Then x and y are

1. in $S(d_2)$ since $S(d_1) \subseteq S(d_2)$, and

2. joined by a tie if and only if they are joined by a tie in d_2.

Therefore, if x and y are joined by a tie in d_1, $\Psi'(x) = \Psi'(y)$ by $\mathit{form}(d_2)$. Similarly, if x and y are not joined by a tie in d_1, we know $\Psi'(x) \neq \Psi'(y)$ by $\mathit{form}(d_2)$. Hence, the spiders distinctness conditions holds for d_1 since it holds for d_2. The sub-diagram relation further states that, for each zone z which is shaded in d_1, the set of spiders $S(z, d_1)$ contains a full set of equivalence class representatives for τ_{d_2} with regard to $S(z, d_2)$. That is, if $y \in S(z, d_2)$ then either $y \in S(z, d_1)$ or there is a spider x in $S(z, d_1) \cap S(z, d_2)$ where x and y are joined by a tie in d_2. Thus, the shaded zones condition is true in d_1 since it is true in d_2. The arrows condition for d_1 is implied by that of d_2 since $A(d_1) \subseteq A(d_2)$. Therefore, I' is valid for d_1 if it is valid for d_2, and $\mathit{form}(d_2) \Rightarrow \mathit{form}(d_1)$ as required.

□

The next result concerning the sub-diagram relation shows that, given a diagram, D, containing nodes n_1 and n_2, labelled by unitary diagrams d_1 and d_2 respectively, if $d_1 \subseteq_S d_2$ but $d_1 \not\subseteq d_2$, then D is unsatisfiable.

Lemma 4.5.2. Let D be a linear generalized diagram that contains nodes, n_1 and n_2, such that the following is true:

1. n_1 is the immediate diagram-labelled ancestor of n_2,

2. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2, respectively, and

3. $d_1 \subseteq_S d_2$.

If $d_1 \not\subseteq d_2$ then D is unsatisfiable.

Proof. Assume that $d_1 \not\subseteq d_2$ and that D is satisfiable; we will show that this is a contradiction. Let $I = (U, \Psi, \Phi)$ be a model for D with valid extension $I' = (U, \Psi', \Phi)$. Since $d_1 \not\subseteq d_2$, we know by the definition of sub-diagrams that one or more of the following is true:

1. there is a spider, x, in $S(d_1)$ such that $\eta_{d_1}(x) \neq \eta_{d_2}(x)$,

2. there is zone, z, which is shaded in d_1 and which is the habitat of a spider, x, in d_2 where x is missing from d_1 and, in d_2, x is not joined by a tie to any spider, y, where y is present in d_1, or

3. there is a pair of spiders $x, y \in S(d_1)$ that are joined by a tie in d_1 and not in d_2, or that are not joined by a tie in d_1 and are joined by a tie in d_2:

$$\exists x, y \in S(d_1)\,((x,y) \in \tau_{d_1} \wedge (x,y) \notin \tau_{d_2}) \vee ((x,y) \notin \tau_{d_1} \wedge (x,y) \in \tau_{d_2}).$$

We will show that, if one of the three conditions is true, $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ is false. Assume that (1) is true. Then there is a spider, say x, in $S(d_1)$, where $\eta_{d_1}(x) \neq \eta_{d_2}(x)$. Let $\eta_{d_1}(x) = z_1$ and $\eta_{d_1}(x) = z_2$. Since $z_1 \neq z_2$ and since, semantically, distinct zones represent distinct subsets of the universal domain, it must be that $\Psi'(z_1) \cap \Psi'(z_2) = \emptyset$. Thus, $\Psi'(x) \subseteq \Psi'(\eta_{d_1}(x))$ implies $\Psi'(x) \not\subseteq \Psi'(\eta_{d_2}(x))$, and the spiders' habitat condition cannot be true in $\mathit{form}(d_1)$ and $\mathit{form}(d_2)$ simultaneously. Thus $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ is false.

Alternatively, assume that (2) is true. Let z be a zone in $Z^*(d_2)$, which contains a spider, x, where $x \in S(z, d_2) - S(z, d_1)$ and where there is no spider $y \in S(z, d_1) \cap S(z, d_2)$ such that $(x, y) \in \tau_{d_2}$. That is, z is shaded in d_2, x inhabits z in d_2 but is missing from d_1, and x is not joined by a tie to any spider which is present in d_1. Given these facts, then if the spiders' distinctness condition is true for d_1 and d_2, the following must be true:

$$\Big(\bigcup_{x\in S(z,d_1)} \Psi'(x)\Big) \neq \Big(\bigcup_{x\in S(z,d_2)} \Psi'(x)\Big). \tag{4.34}$$

Since $d_1 \subseteq_S d_2$, $Z^*(d_1) \subseteq Z^*(d_2)$ and so z is shaded in both unitary diagrams. The shaded zones condition for d_1 includes the sub-formula for z:

$$\Psi'(z) = \bigcup_{x\in S(z,d_1)} \Psi'(x).$$

Equally, the shaded zones condition for d_2 states the following:

$$\Psi'(z) = \bigcup_{x\in S(z,d_2)} \Psi'(x).$$

By (4.34), we know that the shaded zones condition cannot be true for d_1 and d_2 at the same time, and so $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ is false.

Finally, assume (3) is true. Then there is a pair of spiders, say x and y, which are distinct spiders in $S(d_1)$ and are joined by a tie in $S(d_2)$, or vice versa. If x and y are distinct spiders in d_1 then they represent distinct elements: $\mathit{form}(d_1)$ contains the sub-formula $\Psi'(x) \neq \Psi'(y)$. But we know that x and y are joined by a tie in d_2, so $\Psi'(x) = \Psi'(y)$, which is a contradiction. Alternatively, x and y are joined by a tie in d_1 but not in d_2. Here we have $\Psi'(x) = \Psi'(y)$ in $\mathit{form}(d_1)$ and $\Psi'(x) \neq \Psi'(y)$ in $\mathit{form}(d_2)$, again reaching a contradiction. Thus the spiders' distinctness condition for d_1 cannot be true.

Since one or more of conditions (1), (2) and (3) is true, it must be the case that $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ is false in any extended interpretation. We know that D is a linear diagram, and thus that the formula for D is a conjunction that includes the sub-formula $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$, which reduces to $\perp$. Therefore, $\mathit{form}(D)$ is false and we have shown that D is unsatisfiable as required.

□

Theorem 4.5.1: Rule 19, relabel consistent sub-diagram, is valid. Let D_1 be a linear generalized diagram that contains nodes n_1 and n_2 such that the following is true:

1. n_2 is the immediate diagram-labelled descendant of n_1,

2. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2 respectively, and

3. $d_1 \subseteq d_2$.

Let D_2 be the diagram obtained by using the rule to label n_1 in D_1 by $\top$. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. We will show that the formula for D_2 is true under I' whenever the formula for D_1 is true under I'. Diagram D_2 is a copy of D_1 in which n_1 is relabelled by the trivial diagram. That is, the formula for D_1 contains the sub-formula $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ which is replaced in the formula for D_2 by $\top \wedge \mathit{form}(d_2)$. Clearly, $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$ implies $\top \wedge \mathit{form}(d_2)$, and so the formula for D_2 is true whenever the formula for D_1 is true. Thus, $D_1 \models D_2$.

We now show the reverse. We know that $d_1 \subseteq d_2$. By lemma 4.5.1, $\mathit{form}(d_2) \Rightarrow \mathit{form}(d_1)$ and so $\top \wedge \mathit{form}(d_2)$ implies $\mathit{form}(d_1) \wedge \mathit{form}(d_2)$. Thus, the formula for D_1 is true under I' if the formula for D_2 is true under I' and it follows that $D_2 \models D_1$ and $D_1 \equiv_{\models} D_2$ as required. $\square$

Theorem 4.5.2: Rule 20, relabel inconsistent sub-diagram, is valid. Let D_1 be a linear generalized diagram in pushed syntax normal form, containing nodes n_1 and n_2 which satisfy the following:

1. n_1 and n_2 are labelled by the unitary diagrams d_1 and d_2, respectively,

2. n_1 is the immediate diagram-labelled ancestor of n_2, and

3. $d_1 \nsubseteq d_2$.

Let D_2 be the unitary diagram obtained by using rule 20 to transform D_1. Then $D_1 \equiv_{\models} D_2$.

Proof. By lemma 4.5.2, D_1 is unsatisfiable, and so $\mathit{form}(D_1)$ is false under any interpretation. The same is true of D_2, and since neither D_1 or D_2 have any models, $D_1 \equiv_{\models} D_2$ as required. $\square$

Theorem 4.5.3: Rule 21, remove root node, is valid. Let D_1 be a linear generalized diagram which contains nodes n_1 and n_2 such that the following is true:

1. n_1 is the root node of D_1 and is labelled by $\top$,

2. n_2 is the immediate diagram-labelled descendant of n_1.

Let D_2 be the diagram obtained by using the rule to remove n_1 from D_1. Then $D_1 \equiv_{\models} D_2$.

Proof. Let $I = (U\Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. The formula for D_1 consists of the formula for n_1, which is trivially true, conjoined with the formula for D_2. Thus, whenever the formula for D_1 is true under I', the formula for D_2 must also be true under I'. By a similar argument, the formula for D_1 is true under I' whenever the formula for D_2 is true under I'. Thus $D_1 \equiv_{\models} D_2$. □

4.5.3 A decision procedure for linear generalized diagrams

We can use the rules defined and shown to be sound in the previous sections to describe a decision procedure for linear generalized diagrams. The procedure is based on the following algorithm which is used to reduce a linear diagram in pushed syntax normal form to a generalized diagram consisting of a single diagram-labelled node.

Algorithm 4 (Linear diagram in pushed syntax normal form to single diagram-labelled node)**.** Let D_1 be a linear generalized diagram in pushed syntax normal form. Transform D_1 to a generalized diagram consisting of a single diagram-labelled node, D_2, using the following steps:

1. If the root node of D_1 is a leaf node, then let $D_2 = D_1$ and we are done.

2. Otherwise, let n_1 and n_2 be the root node of D_1 and its immediate diagram-labelled descendant, labelled by the unitary diagrams d_1 and d_2 respectively. If $d_1 \not\subseteq d_2$, then the conditions exists to apply rule 20, relabel inconsistent sub-diagram, giving a diagram, D_2, which consists of a single diagram-labelled node labelled by $\perp$. Alternatively, $d_1 \subseteq d_2$ and we are able to use rule 19, relabel sub-diagram, to relabel n_1 by $\top$ in D, giving E_1. Next, we use rule 21, remove root node, to remove n_1 from E_1, giving E_2. We repeat the process at step (1) in the context of E_2.

Next we show that the above algorithm is terminating and produces a diagram which is equivalent in meaning to the original.

Lemma 4.5.3. Let D_1 be a linear generalized diagram in pushed syntax normal form and let D_2 be the diagram obtained by using algorithm 4 to transform D_1. Then the following is true:

1. D_2 consists of a single diagram-labelled node,
2. $D_1 \equiv_{\models} D_2$, and
3. algorithm 4 terminates.

Proof. If algorithm 4 halts at step (1) then $D_1 \equiv_{\models} D_2$ trivially. Let n_1 and n_2 be the root node of D_1 and its immediate diagram-labelled descendant, labelled by the unitary diagrams d_1 and d_2 respectively. In step (2), if $d_1 \not\subseteq d_2$, then we use rule 20, relabel inconsistent sub-diagram, to replace D_1 by D_2, the generalized diagram consisting of a single node labelled by $\perp$. By theorem 4.5.2, $D_1 \equiv_{\models} D_2$. Alternatively, if $d_1 \subseteq d_2$ in step (2), we use rule 19, relabel sub-diagram, to relabel n_1 by $\top$ in D, giving E_1. By theorem 4.5.1, $D_1 \equiv_{\models} E_1$. Next, we use rule 21, remove root node, to remove n_1 from E_1, giving E_2. By theorem 4.5.3, $E_1 \equiv_{\models} E_2$ and so $D_1 \equiv_{\models} E_2$. Each step of the algorithm either halts the process or reduces the number of nodes in D_1. Thus, since D_1 is finite, the process is terminating, results in a diagram consisting of a single diagram-labelled node, and $D_1 \equiv_{\models} D_2$ as required. □

To close this section we state the conditions under which a linear diagram in pushed syntax normal form, D, is satisfiable; D is satisfiable if and only if D can be transformed into a diagram that consists of a single node labelled by a satisfiable unitary diagram. First, we show that a linear diagram containing an unsatisfiable unitary diagram is itself unsatisfiable.

Lemma 4.5.4. Let $D = (V, W, E, l)$ be a linear generalized diagram containing a node, n, labelled by a unitary diagram, d. If d is unsatisfiable then D is unsatisfiable.

Proof. Let $I = (U, \Psi, \Phi)$ be an interpretation with extension $I' = (U, \Psi', \Phi)$. Assume that d is unsatisfiable. The formula for D is as follows:

$$\mathit{form}(D) = \bigwedge_{n \in V} \mathit{form}(l(n)). \tag{4.35}$$

We know that $n \in V$, $l(n) = d$ and $\mathit{form}(l(n)) = \bot$, since d is unsatisfiable. Therefore (4.35) reduces to $\bot$ and D is unsatisfiable. □

Lemma 4.5.5. Let D_1 be a linear generalized diagram in pushed syntax normal form and let D_2 be the diagram obtained by using algorithm 4 to transform D_1. Then D_1 is satisfiable if and only if D_2 is satisfiable.

Proof. Assume that D_1 is satisfiable. By lemma 4.5.3, $D_1 \equiv_{\models} D_2$ and so D_2 is satisfiable. Next, assume that D_2 is unsatisfiable. Since D_2 was obtained by using algorithm 4 to transform D_1, one of the following must be true:

1. D_1 contains an unsatisfiable unitary diagram, or
2. D_1 contains diagram-labelled nodes n_1 and n_2 such that the following is true:
   (a) n_1 is the immediate diagram-labelled ancestor of n_2,
   (b) n_1 and n_2 are labelled by unitary diagrams d_1 and d_2 respectively, and
   (c) $d_1 \not\subseteq d_2$.

If the first condition is true, then D_1 is unsatisfiable by lemma 4.5.4. If the second condition is true, then D_1 is unsatisfiable by lemma 4.5.2. Thus, if D_2 is unsatisfiable then D_1 is also unsatisfiable. Hence we have shown that D_1 is satisfiable if and only if D_2 is satisfiable. □

4.6 The decision procedure

We now have the results we need to describe the decision procedure for the existential fragment. For convenience, we combine the algorithms relating to normal forms given earlier in this chapter to form an algorithm which transforms a generalized diagram into disjunctive normal form in one step. In Figure 4.78, the dashed box illustrates the way that the new algorithm combines the steps required to convert a diagram from the existential fragment to a diagram in disjunctive normal form.

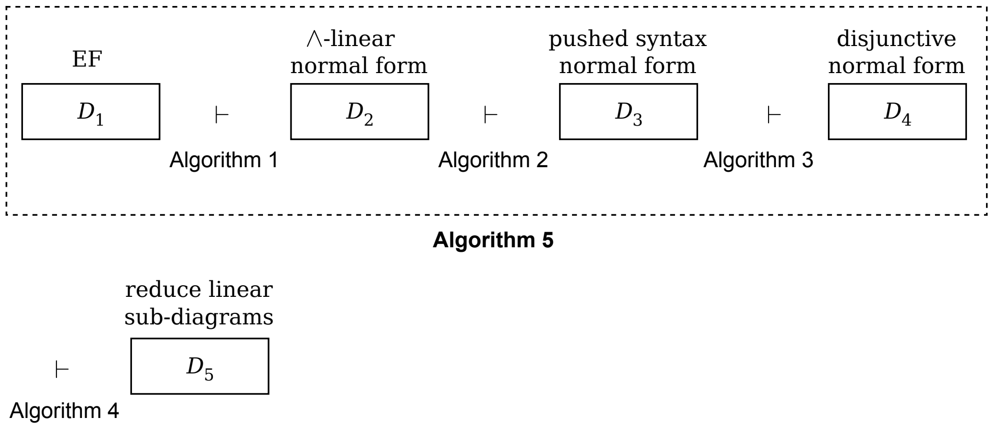


Figure 4.78: The algorithm to convert a generalized diagram to disjunctive normal form.

Algorithm 5 (Generalized diagram to disjunctive normal form)**.** Let D_1 be a generalized diagram. Transform D_1 into a second diagram, D_2, using the following steps:

1. Use algorithm 1, generalized diagram to $\wedge$-linear normal form, to transform D_1, giving E_1.

2. Use algorithm 2, $\wedge$-linear to pushed syntax normal form, to transform E_1, giving E_2.

3. Use algorithm 3, pushed syntax to disjunctive normal form, to transform E_2, giving D_2.

We show that the above algorithm terminates, and results in a diagram which is equivalent to the original diagram, and is in each of the normal forms.

Theorem 4.6.1: Let D_1 be a generalized diagram and let D_2 be the diagram obtained by using algorithm 5 to transform D_1. Then the following is true:

1. D_2 is in $\wedge$-linear, pushed syntax and disjunctive normal forms,

2. $D_1 \equiv_{\models} D_2$, and
3. algorithm 5 terminates.

Proof. This follows from theorems 4.2.2, 4.3.11 and 4.4.5. □

After using algorithm 5 to convert a generalized diagram, D_1, to an equivalent diagram in disjunctive normal form, D_2, we examine the set of linear branches of D_2, $\mathcal{B}$. For each branch $B \in \mathcal{B}$, we use algorithm 4, which reduces linear generalized diagrams, to reduce B to a generalized diagram, B', consisting of a single node. Then, either B' is labelled by $\perp$, in which case B' and hence B are unsatisfiable, or B' is labelled by a unitary diagram $b' \neq \perp$. In this case, we appeal to the decision procedure for unitary diagrams given in chapter 3. These steps form the decision procedure for the existential fragment, which we use as the basis for the main theorem of this chapter; a diagram from the existential fragment, D, is satisfiable if and only if, after transforming D to a diagram in disjunctive normal form with a set of disjunctive branches, $\mathcal{B}$, some element of $\mathcal{B}$ can be reduced to a satisfiable unitary diagram.

Theorem 4.6.2: The existential fragment is decidable.

Proof. Let D_1 be a diagram drawn from the existential fragment. By theorem 4.6.1, we can use algorithm 5 to convert D_1 in finite steps to a diagram, D_2, where D_2 is in $\wedge$-linear, pushed syntax and disjunctive normal form, and $D_1 \equiv_{\models} D_2$. If D_2 is a linear diagram, then by lemma 4.5.3, we can use algorithm 4 to reduce D_2 to a generalized diagram, D_3, consisting of a single node and where $D_2 \equiv_{\models} D_3$. Let d_3 be the unitary diagram labelling the root node of D_3. Clearly, $D_3 \equiv_{\models} d_3$. By theorem 3.3.5, we can determine the satisfiability of d_3. By lemma 4.5.5 and since $D_1 \equiv_{\models} D_2 \equiv_{\models} D_3 \equiv_{\models} d_3$, D_1 is satisfiable if and only if d_3 is satisfiable. There is a decision procedure for determining the satisfiability of d_3, so there is such a procedure for D_3 and we are done.

If D_2 is not a linear diagram, let $n_\vee$ be the (unique) $\vee$-labelled node in D_2 and let $\mathcal{B}$ be the set of sub-diagrams of D_2 induced by the immediate descendants of $n_\vee$. Let $\mathcal{B}'$ be the set obtained by applying algorithm 4 to each of the sub-diagrams of D_2 in the set $\mathcal{B}$. Then by lemma 4.5.3, the following holds for any interpretation:

$$\Big(\bigvee_{B\in\mathcal{B}} \mathit{form}(B)\Big) \equiv \Big(\bigvee_{B'\in\mathcal{B}'} \mathit{form}(B')\Big). \tag{4.36}$$

Each generalized diagram in $\mathcal{B}'$ consists of a single diagram-labelled node. Let $B' \in \mathcal{B}'$ and let b' be the unitary diagram labelling the root node of B'. By theorem 3.3.5, we can determine the satisfiability of b', and thus B'. Since $B' \equiv_{\models} b'$, B' is satisfiable if and only if b' is satisfiable. There is a decision procedure for the satisfiability of b', and so there is one for B'. Since $\mathcal{B}'$ is finite, we have a decision procedure for determining whether $\mathcal{B}'$ contains a satisfiable diagram. By (4.36), we can determine the same for $\mathcal{B}$. Hence, we can determine the satisfiability of D_2. Since $D_1 \equiv_{\models} D_2$, D_1 is satisfiable if and only if D_2 is satisfiable, and the existential fragment is decidable.

□

In this chapter we have constructed a set of algorithms, based on syntactic operations and normal forms, that can be used to judge the satisfiability of a generalized diagram. Theorem 4.6.2, which shows that the existential fragment is decidable, concludes the main part of the thesis.

Chapter 5

Conclusions and further work

We begin by summarising the results and the contribution of the thesis, before discussing several promising areas of further work in which our work can be extended.

5.1 Results and contribution of this thesis

After an examination of the manner in which syntax interacts to give rise to inconsistency in generalized unitary diagrams, we defined the class of unitary γ-diagrams. In this class, the consistency of a unitary diagram can be judged by an inspection of its spiders and arrows. We developed standard interpretations for γ-diagrams, building on work in [38] which, in turn, is analogous to the canonical models used in the classical decision problem. Making use of standard interpretations, we established the relationship between our definitions of consistency and satisfiability. Using γ-diagrams as a reduction class for the unitary fragment, we extended this result to develop a decision procedure for all unitary diagrams.

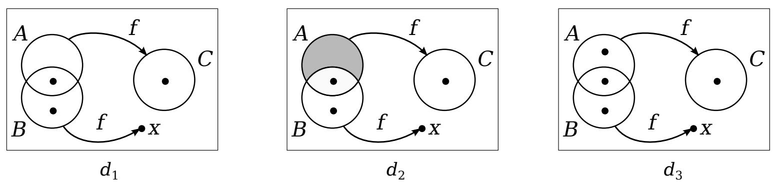


Figure 5.1: A unitary diagram and its γ-components.

To recap the unitary decision procedure, consider Figure 5.1, in which d_1

is a diagram from our unitary existential fragment. Diagram d_1 is not a γ-diagram since there is a zone, $(\{A\},\{B,C\})$, which is unshaded and contains no spiders. We cannot determine the consistency of d_1 simply by inspecting its spiders and arrows (that is, by using the method devised in this thesis). Diagrams d_2 and d_3 are the γ-components of d_1, in which the zone $(\{A\},\{B,C\})$ is either shaded or contains a spider, and theorem 3.3.4 establishes that d_1 is equivalent to the disjunction of d_2 and d_3. Taking two disjoint sets of the arrows of d_2, $X = \{(f,A,C)\}$ and $Y = \{(f,B,x)\}$, we can see that if we subtract the target spiders of Y from the target spiders of X, there are spiders remaining. If we subtract the source spiders of Y from the source spiders of X, however, there are no spiders remaining. By definition 3.1.4, diagram d_2 is inconsistent, and, by theorem 3.1.1, unsatisfiable. Carrying out the same process with diagram d_3, we can see that it is satisfiable. Since $d_1 \equiv_{\models} d_2 \vee d_3$ and d_3 is satisfiable, we can determine that d_1 is satisfiable. A simplified version of the contents of chapter 3 was published in [4].

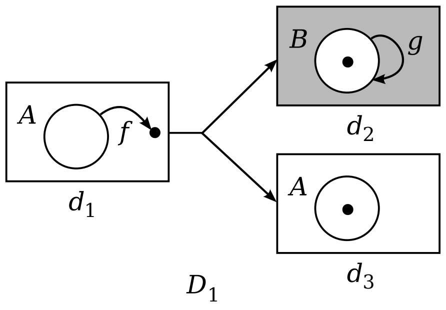


Figure 5.2: A non-unitary generalized diagram.

In chapter 4, we extend the decision procedure to the entire existential fragment. We developed a series of normal forms that allow us to transform a generalized diagram such as D_1, Figure 5.2, into a form whereby we can determine its consistency. In particular, we showed that we can linearise $\wedge$-labelled nodes to give $\wedge$-linear normal form, push syntax forwards through a diagram to give pushed syntax normal form, and remove nested $\vee$-labelled nodes to give disjunctive normal form. We showed that transforming a diagram through this sequence of the normal forms produces a sequence of semantically equivalent diagrams, resulting in one whose meaning is contained in a set of unitary diagrams. These results, in combination with the decision procedure for unitary diagrams, allow us to describe a decision procedure for the entire system. In Figure 5.3, diagram D_n shows the result of applying this procedure to the diagram in Figure 5.2. We can determine the consistency of D_n by applying the unitary procedure of chapter 3 to the unitary diagrams labelling its leaf nodes.

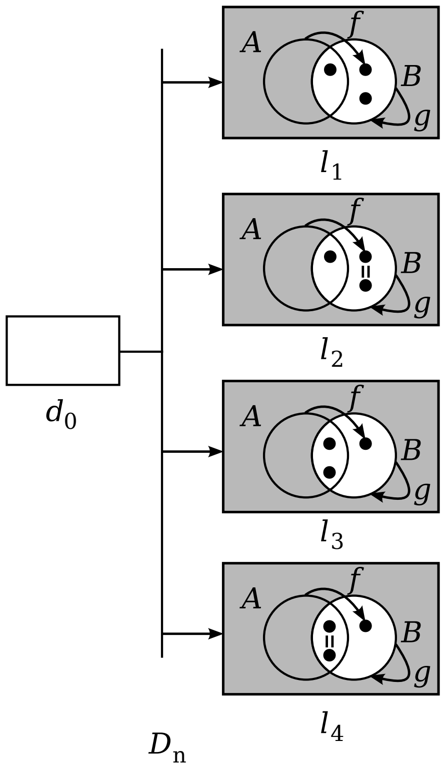


Figure 5.3: A generalized diagram in disjunctive normal form.

As we discussed in section 1.1.2, the decision problem for the more expressive diagrammatic logics, such as constraint diagrams, has not yet been widely studied. Our study of the ways in which syntax interacts to give rise to inconsistency in generalized unitary diagrams adds to the knowledge of the Euler/Venn family of notations. This study leads to the decision procedure for unitary diagrams, which was our first main contribution. It is interesting, although perhaps unsurprising, that our unitary decision procedure depends on techniques first developed to tackle the decision problem in symbolic logic, such as canonical models and reduction classes, although we use these strategies in a diagrammatic context.

The tree structure of generalized diagrams is novel within the Euler/Venn family; our decision procedure for the non-unitary case consists of sound manipulations of that tree structure, as well as sound manipulations of the content of unitary diagrams labelling nodes in the tree. The decision procedure for the full existential fragment was our second main contribution.

The fragment of GCDs we study was selected to be decidable. Although the expressiveness of our fragment is an open problem, we conjecture that the fragment is strictly less expressive than the Bernays-Schönfinkel-Ramsey class with equality, which consists of sentences with two place predicates and a $\exists^*\forall^*$ prefix [2]. This seems to be the case because translating the semantic formula of a generalized constraint diagram in the existential fragment to first order logic yields a sentence in which all $\exists$ come before $\forall$, but there are formulae with the prefix $\exists^*\forall^*$ which cannot be expressed as diagrams in the EF, such as $\exists x \forall y\, f(y, x)$. It would be possible to translate each diagram to a sentence in symbolic language then use an existing decision procedure; an efficient decision procedure for this class of formulae, building on techniques used to solve the Boolean satisfiability problem, is given in [9]. The benefit of the current work, however, is found in the focus on the diagrammatic nature of the logic in question. Symbolic logic decision procedures give no real insight into how unsatisfiability arises in diagrammatic notations, something which is of key interest to users of the notations.

Although our fragment is a restriction of the full system of GCDs, our results could be of practical benefit to users of the whole system. As a step towards verifying the consistency of a model, it would be possible to develop software tools which implement partial model checking, used to judge the consistency of that part of a model which does not include universal spiders, and to identify inconsistency arising solely from the presence or absence of existential spiders in any model. This information is of fundamental importance to those using the notation to model, for instance, software systems. We discuss the implications of implementing the decision procedure in the next section.

Unlike earlier work, the inference rules in our system are based on purely syntactic transformations. We believe that there are a number of benefits to be gained from this approach, which we have previously explored in [3]. The use of transformations leads to more concise definitions of rules and to the benefits of modularity, whereby a transformation is defined once and used in the definition of several rules of inference. Furthermore, we have defined transformations with a level of generality that allows them to be used in a wide range of reasoning contexts. For example, the add contour transformation (see page 82) is parametrised with the sets of zones which will be either inside or outside of the new contour, called Z_{in} and Z_{out} respectively, and with the sets of spiders which will be either inside or outside of the new contour, S_{in} and S_{out}. In [38], an *add contour* inference rule is defined so that the new contour splits every zone in the premise diagram. We define rule 6, add contour, to have this effect by assigning $Z_{in} = Z_{out} = C(d)$. Figure 5.4 shows an example of this choice of Z_{in} and Z_{out}.

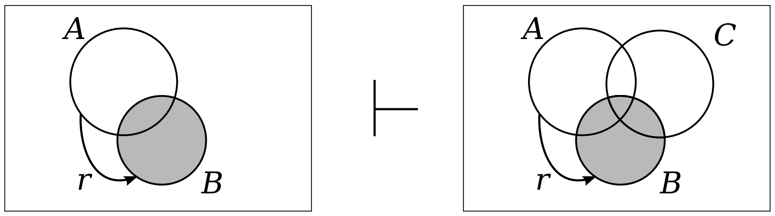


Figure 5.4: Adding a contour so as to split every zone.

In the case of rule 2, add contour over spider, we define a valid rule which adds a contour so that it splits a single zone, which is the habitat of a particular spider (see Figure 5.5). In the definition of the rule, we achieve this effect using the same add contour transformation with a different choice of Z_{in} and Z_{out}, demonstrating the flexibility of the transformations approach.

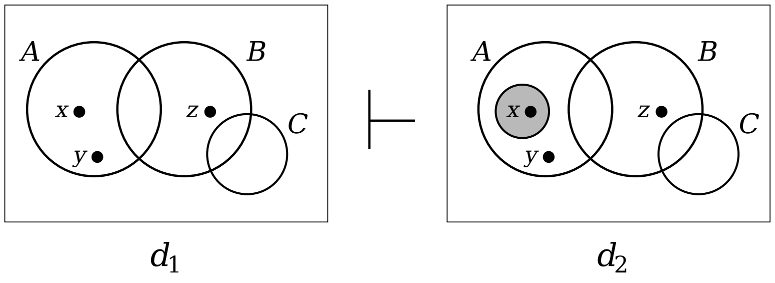


Figure 5.5: Adding a contour without splitting every zone.

In summary, we have devised novel decision procedures for γ-diagrams, unitary diagrams and compound diagrams from the existential fragment of GCDs. These decision procedures provide insight into how the diagrammatic syntax gives rise to inconsistent formations. In addition we have devised a set of sound inference rules that can be used to transform diagrams into various normal forms, which could well be of use when considering the issue of completeness, further discussed below.

5.2 Further work

Finally, we highlight and discuss areas for further work. Some of these areas are directly supported by the contribution of our thesis, such as the development of a completeness proof, and some are more indirectly related, such as our discussion of usability.

5.2.1 The relationship between the existential fragment and the full system

Our work presents a system of generalized constraint diagrams which differs from the original presentation in various ways: we exclude universal spiders, resulting in a decidable system, we restrict spiders to have a single foot, so that all unitary diagrams are α-diagrams, and we have added ties between spiders, representing equality. Of these changes, the restriction to α-diagrams and the addition of ties are relatively superficial, and all of our results can be adapted to a system including spiders with multiple feet and excluding ties with little difficulty. Spiders with several feet represent disjunctive information and thus a constraint diagram which contains a spider with several feet is equivalent to a disjunction of α-diagrams [46]. Ties between spiders are a notational convenience which can be removed by altering the syntax. The exclusion of universal spiders, on the other hand, reduces the expressiveness of the system and we would need to make a number of significant changes to accommodate universal spiders.

As well as changes to the syntax, the semantics of our diagrams would be altered by the reintroduction of universal spiders. As in the original presentation [41], the formulae of a diagram would need to introduce new universal quantifiers to quantify over all extended interpretations, as well as the current existential quantification which interprets derived contours and existential spiders. We would also need to reintroduce the invariant found in the original system that each diagram-labelled node which introduces a universal spider can not introduce any other spider. Since the reintroduction of universal spiders would result in semantic formulae with arbitrary quantifier alternation, this would result in an undecidable system.

The original work on generalized constraint diagrams does not discuss inconsistency, but the forms of inconsistency which arise from universal spiders in non-generalized constraint diagrams are described in [38]. Figure 5.6 shows three examples of inconsistency arising from arrows sourced on universal spiders.

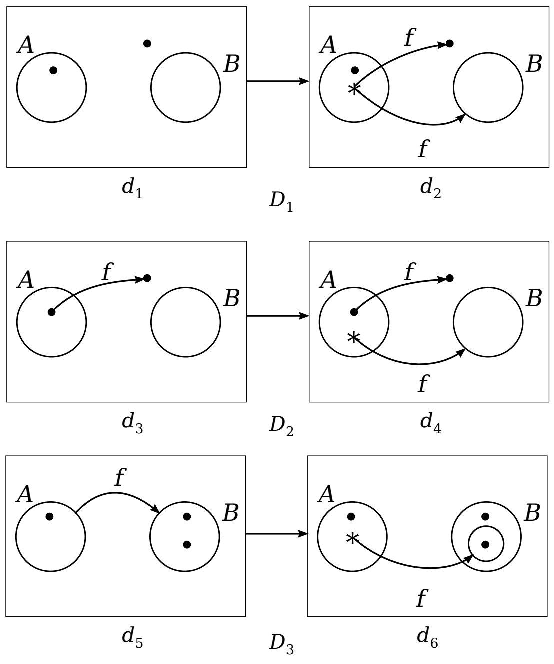


Figure 5.6: Inconsistency arising from arrows sourced on universal spiders.

The type of inconsistency found in diagrams D_1 and D_2, Figure 5.6, extends the notion of target inconsistency, described in section 3.1.1, page 55. In each case, the target of an arrow supplies contradictory information. In D_1, given an extended interpretation (U, Ψ', ϕ), the formula for the unitary diagram d_1 is as follows:

$$\exists x \exists y (PTC(d_1) \wedge SZC(d_1) \wedge (\Psi'(x) \subseteq \Psi'(\{A\}, \{B\}) \\ \wedge \Psi'(y) \subseteq \Psi'(\emptyset, \{A, B\})) \wedge SDC(d_1) \wedge AC(d_1)), \quad (5.1)$$

where only the spiders habitat condition is given in full. In the formula for d_2, the arrows condition contains the following two clauses:

$$\forall x(x \in S((\{A\},\{B\}),d_2) \Rightarrow \Psi'(\{x\}).\Phi(f) = \Psi'(\{y\})), \tag{5.2}$$

and

$$\forall y(y \in S((\{A\},\{B\}),d_2) \Rightarrow \Psi'(\{y\}).\Phi(f) = \Psi'(B)). \tag{5.3}$$

By the spiders habitat condition in (5.1), we can deduce that there is at least one element in the set $\Psi'(\{A\},\{B\})$, and thus $\Psi'(A)$ is not empty. Also by (5.1), we know that $\Psi'(y) \subseteq \Psi'(\emptyset,\{A,B\})$ and thus, by the definition of Ψ', $\Psi'(y) \nsubseteq \Psi'(B)$. It follows that (5.2) and (5.3) contradict each other. Note that the contradiction depends on the fact that $\Psi'(\{A\},\{B\})$, the mapping for the zone over which the universal spider quantifies, is not empty. Otherwise, both (5.2) and (5.3) are true.

In diagram D_2, Figure 5.6, contradictory information is again supplied by the targets of arrows. The type of inconsistency found in diagram D_3 extends the notion of source inconsistency (section 3.1.1), as there are too few source spiders to satisfy the relation represented by an arrow label. Both source and target consistency are accommodated by our definition of consistency, and we can see that it would be possible to extend the definition to accommodate universal spiders. This extension would enable us to show that inconsistent diagrams containing universal spiders are unsatisfiable. The reverse, to show that unsatisfiable diagrams are inconsistent, would not be possible since the full system is undecidable.

5.2.2 Implementing the decision procedure

Although tools exist for drawing constraint diagrams [15], there are currently no tools for reasoning with them. Were such tools to be developed, their authors could make immediate use of our decision procedure. However, the definitions and algorithms developed throughout our work were developed without consideration for computational complexity. When implementing the decision procedure, we need to consider performance, complexity, and the possibility of creating an optimised procedure. The task of judging the satisfiability of a unitary γ diagram has deterministic exponential complexity. For example, consider Figure 5.7, diagram

d. Let X be the set of arrows in d:

$$X = \{(f, x, B), (f, y, z), (f, A, C)\}.$$

The decision procedure compares source and target spiders for each element of the powerset of X, and its complement. That is, the process examines the sets of arrows $\emptyset$ and X, followed by $\{(f, x, B)\}$ and $\{(f, y, z), (f, A, C)\}$, and so on. If n is the cardinality of $\mathbb{P}X$ then the process takes 2^n steps. In general, this gives an upper bound on the steps required by the procedure, since the process can stop when we encounter inconsistency. The process compares arrows with the same label, and for each arrow label, l, in a unitary diagram, d, the largest number of steps is determined by, and is exponential in, $|\mathbb{P}A(l, d)|$.

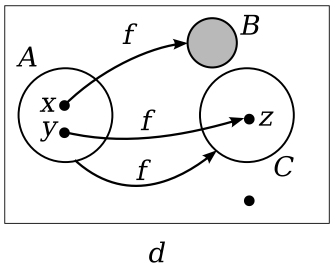


Figure 5.7: The complexity of the unitary decision procedure.

Several of the other steps in the unitary decision procedure, such as finding the γ components of an α diagram, are similarly computationally expensive. Such complexity is typical for a decision procedure of this type. For example, in [2], Börger et al. describe a decision procedure for the Bernays-Schönfinkel-Ramsey class. They show that a satisfiable sentence in this class has a model, m, with a finite domain, U, give a constructive method for finding m, and show that the complexity of the method is exponential in the cardinality of U [2, p259]. The decision procedure for (non-unitary) generalized diagrams also has deterministic exponential complexity. In this case, however, there are various *ad hoc* optimisations which could be used to improve the performance of the process in real time, such as pruning the graph underlying a generalized diagram at various stages, reducing the number of calculations needed in following stages.

For instance, when pushing contours, we form the set of all add contour components for a unitary diagram (see section 4.3.2). In Figure 5.8, diagrams d_3 and

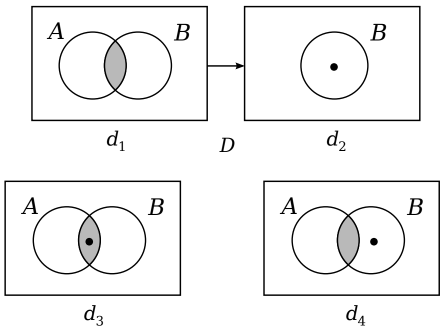


Figure 5.8: Inconsistent branches produced by pushing contours.

d_4 are the add contour components for B in d_2. When pushing B from d_1 to d_2 in the generalized diagram D, our procedure replaces d_2 by the disjunction of d_3 and d_4. However, we can see that d_3 is inconsistent in the context of D: it contains a spider in the zone $(\{A, B\}, \emptyset)$, which is shaded and untouched in d_1. Our procedure would discard the unitary diagram d_3 in the final stage of the decision procedure, that which reduces linear diagrams. In the case of diagrams larger than D, Figure 5.8, removing diagrams such as d_3 immediately after pushing contours would provide a considerable optimisation. The same strategy could be used to reduce the number of add spider with ties components (again, see section 4.3.2).

5.2.3 Usability

In the literature review we discussed the usability of diagrammatic notations, and the rôle played by usability in the development of generalized constraint diagrams. We noted that the usability of constraint diagrams, as opposed to their underlying Euler diagram notation, requires further study, and highlighted the recent small study in [11]. We also believe it would be profitable to compare the usability of generalized constraint diagrams with that of the non-generalized notation. In [8], the author, in collaboration with Coppin and Hockema, conjectured that certain properties of a notation may make the notation effective for communicating proofs, and showed that these properties are held by generalized constraint diagrams. Cognitive factors are central to the adoption of a notation by users, and much further investigation is required to establish the ways in which these factors interact, given a notation and a specific task. We believe it would be possible to identify the effect of the features distinctive to GCDs by conducting a study in which software engineers undertake a comprehension task using GCDs and non-generalized constraint diagrams.

5.2.4 Completeness

The next natural step in the metatheory of generalized constraint diagrams is to show the completeness of a set of inference rules. In this section we discuss this goal and possible strategies for achieving it. Stapleton [38] showed completeness for a (non-generalized) constraint diagram system. The strategy used is illustrated in Figure 5.9, which is taken from [38, p250]. Given two diagrams, D_1 and D_2, where $D_1 \vDash D_2$, the problem is to show that $D_1 \vdash D_2$. That is, we need to show that we have sufficient (sound) inference rules to transform D_1 into D_2. Stapleton's strategy proceeds as follows: first, manipulate both diagrams until they are transformed into diagrams, D_1^β and D_2^β respectively, where D_1^β and D_2^β have the same zone sets and are disjunctions of β-diagrams. In Figure 5.9 this process is covered by steps 1 to 5. In the process of building our decision procedures, we have provided sound inference rules which accomplish the generalized version of each of the steps. Thus, diagrams D_1^β and D_2^β have a structure which is the non-generalized equivalent of the disjunctive normal form presented in chapter 4.

The next step in this strategy is to add syntax to D_1^β, to produce its *maximal form*, say D_1', where $D_1 \equiv_\vdash D_1'$. Next, remove syntax from D_2^β to produce its *minimal form*, say D_2', where $D_2 \equiv_\vdash D_2'$. Then we need to show that we can embed D_2' in D_1' in some sense, possibly by showing that we can remove syntax from D_1' until it is isomorphic to D_2'.

We have begun to investigate the use of this strategy for generalized constraint diagrams, developing the notion of *potential* and *redundant* syntax for GCDs, although this preliminary work is not in the thesis. To give a flavour of the task, we describe the task of finding potential shading and redundant arrows.

Potential syntax is that which can be added to a diagram without changing its meaning. For instance, in the diagram in Figure 5.10, we can add shading to the zones $(\{B\}, \{A, C\})$ and $(\{C\}, \{A, B\})$ without changing the meaning of the

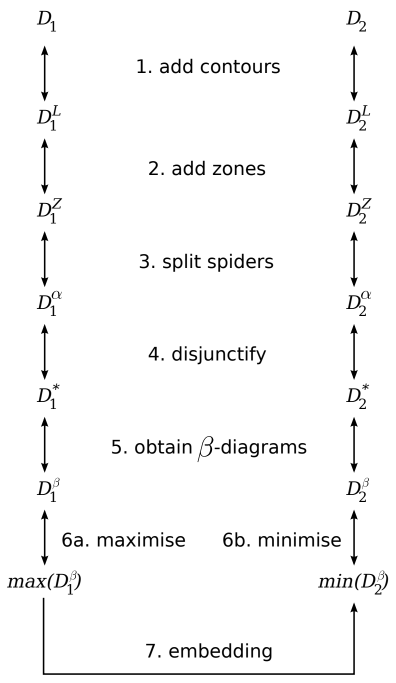


Figure 5.9: Stapleton's completeness proof strategy.

diagram; we say that these zones are shadeable. We can infer their shadeability as follows: the arrow (f, A, B) tells us that, when the domain of the relation represented by f is restricted to the empty set, then the image is the set represented by B. It must be true that the relational image of f is the empty set in this case, and so we can infer that the set B represents is empty in any model. Thus, we can describe a valid inference rule which adds shading to the zone inside B. The same reasoning allows us to shade the zone inside C.

This form of shadeability, which arises from the relative placement of arrows, is the only form of shadeability found in constraint diagram systems without derived contours. Informally, we can see that, in the semantics, we need positive evidence to infer the cardinality of a given set. Derived contours, however, represent arbitrary subsets, and their semantics is given by asserting the existence of an extended interpretation. Thus, we can restrict their cardinality by adding shading at any time, providing this does not change the meaning of the underlying diagram in which they are placed.

In Figure 5.11, diagram d_1, c_1 is a derived contour. If (U, Ψ, Φ) is an interpretation for d_1, the zone $(\{c_1\}, \{A\})$ represents an unnamed subset of $U - \Psi(A)$

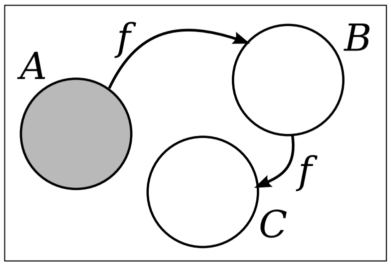


Figure 5.10: Shadeable zones.

which contains at least one element. Clearly, there exists some extension of Ψ, Ψ', where $\Psi'(\{c_1\},\{A\})$ contains exactly one element, and thus $(\{c_1\},\{A\})$ is shadeable. Any of the other zones in the diagram is shadeable by the same reasoning, but adding shading to certain zones limits the shadeability of the remaining zones. In d_2, the zones $(\{A\},\{c_1\})$ and $(\{c_1\},\{A\})$ are shaded, and $d_1 \vDash d_2$. To add shading to $(\{A,c_1\},\emptyset)$ in d_2 would be to claim that the set represented by A contains exactly two elements, something which is not true for all interpretations. Since one zone inside A is left unshaded in d_2, any model for d_1 is a model for d_2. We say that the diagrams d_2 to d_5 are the *maximally shaded* forms of d_1. Adding potential shading to a diagram may therefore give rise to choice and leads, in general, to a set of maximally shaded diagrams. This is the first difficulty in adapting Stapleton's strategy to the generalized case: there is no unique maximal form of d_1.

Next, we consider the task of finding the minimal form of a generalized unitary diagram, which is achieved by removing redundant syntax. Redundant syntax is that which can be removed without changing the meaning of a diagram. The presence of derived contours increases the ways in which syntax can be considered redundant. In Figure 5.12 we can infer that (f,s_2,t) is redundant since (f,s_1,t) tells us, informally, that some element of s_1 is related to the element represented by the spider inside t. In any model for the diagram in Figure 5.12, the only elements available to be related to the element represented by the spider in t are those in s_2, as all other elements in s_1 are in the source of an arrow labelled f which targets the empty set. The arrow (f,s_1,t) is redundant by a similar argument.

When constructing the minimal form of a unitary diagram, the goal is to show

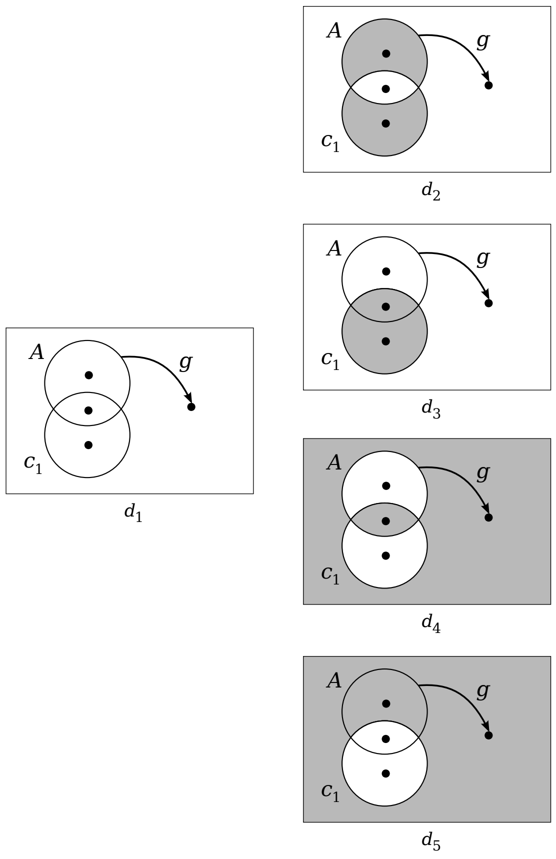


Figure 5.11: Shadeable zones and derived contours.

that the semantic and syntactic notions of redundancy for arrows coincide. The syntactic notion of redundancy is the set of syntactic conditions that allow us to identify when an arrow is redundant. The conditions relate to the presence of a set of arrows; intuitively, we can see that if an arrow (l, s, t) is a semantic redundant arrow for a unitary γ-diagram d, there must be a set of arrows in $A(d) - \{(l, s, t)\}$ which allows us to infer this fact.

In Figure 5.13 we can infer that (f, s_1, t) is redundant due to the arrows sourced on s_2 and s_3, neither of which is redundant itself. It is easier to see that this is true if we remove certain zones from the diagram.

In Figure 5.14, d_1 is the diagram obtained by removing all but one of the untouched and shaded zones from the diagram in Figure 5.13. In d_2 the arrow (f, s_1, t) is removed, and it is easy to see that the arrows of d_2 provide the same information as those of d_1. Diagram d_3 is obtained by removing the arrow (f, s_3, t)

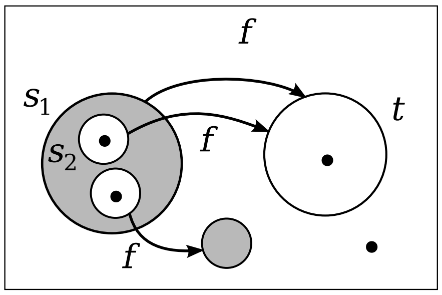


Figure 5.12: A redundant arrow.

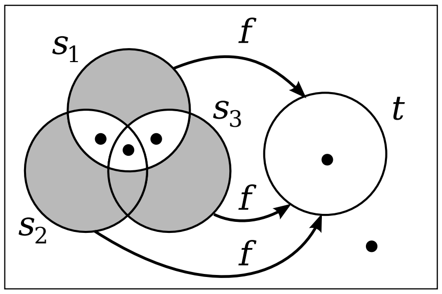


Figure 5.13: Redundancy inferred from a set of arrows.

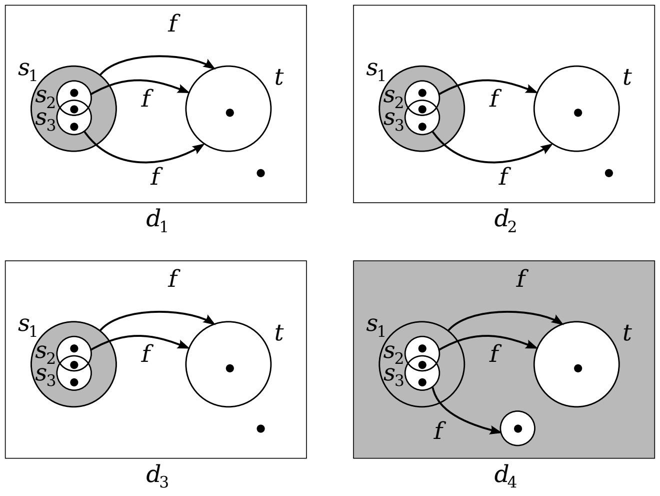


Figure 5.14: Redundancy inferred from a set of arrows continued.

from d_1. In this case we have weakened the information provided by the diagram, as it is no longer necessary that elements inside s_3 be related to anything under f. Thus, (f, s_3, t) is not a redundant arrow. Diagram d_4 (to which shading is added so that d_4 is a γ-diagram) illustrates this, since it is satisfiable and entails d_3 but not d_1.

So far we have seen examples in which an arrow's redundancy is inferred from a set of arrows whose source spiders are equal to those of the redundant arrow; in general, this need not be the case. In Figure 5.15 the arrow (f, s_2, t) in diagram d_1 is redundant. The arrow (f, s_1, t) asserts that the single element in t is related to at least one element of s_1 under f; (f, s_2, t) asserts that the element in t is related to one or more elements of s_2 under f.

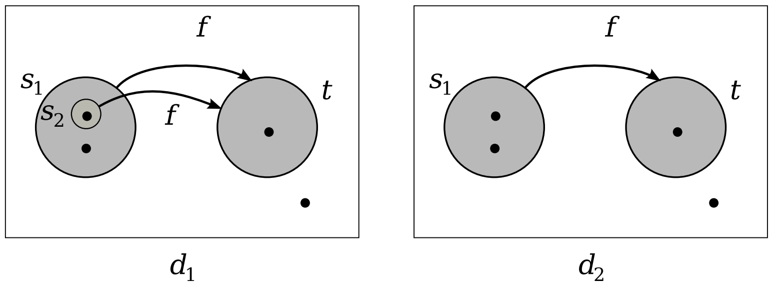


Figure 5.15: Redundancy and the sources of arrows.

The information provided by (f, s_2, t) is thus, in some sense, a subset of that provided by (f, s_1, t). The redundancy of (f, s_2, t) depends on two further conditions: the fact that all spiders within s_1 in $d_1 - \{(f, s_2, t)\}$ are *indistinguishable modulo derived contours*, and the fact that t contains exactly one spider. Two pieces of syntax x and y in a unitary diagram d can be considered indistinguishable if there is an automorphism on d (a structure-preserving mapping from d to d) that maps x to y. After removing the derived contour, s_2, from diagram d_1, Figure 5.15, as is done in diagram d_2, the two spiders inside s_1 are indistinguishable, and so we say that the spiders in s_1, diagram d_1, are indistinguishable modulo derived contours. Thus, when identifying redundant syntax in generalized unitary diagrams, derived contours give rise to the need to consider indistinguishable syntax.

In Figure 5.16, diagram d_1, the derived contours are labelled s_1, s_2, s_3 and t. The arrow (f, s_2, t) is redundant. We can infer this by combining the information provided by the other arrows in the diagram, (f, s_1, t) and (f, s_3, t). Diagram d_2 shows d_1 with the redundant arrow removed. To mix syntax and semantics temporarily, we can infer that, because of (f, s_3, t), the image of f when the

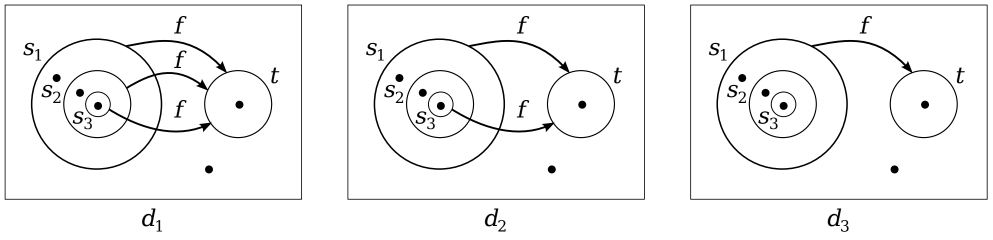


Figure 5.16: Inferring redundancy from a subset of the source spiders.

domain is restricted to s_2 is at least t; because of (f, s_1, t), it as at most t. Diagram d_3 shows d_2 after the removal of (f, s_3, t). Because the only arrow in the diagram gives us no information on which of the spiders of s_1 are related to the spider in t, we have no basis on which to add (f, s_2, t) to d_3. Another way of thinking of this is that if we have three sets A, B and C where $A \supset B \supset C$, and we have a relation R on A, then we know that $A.R \supseteq B.R \supseteq C.R$. It follows that if we know that $A.R = C.R$ then we can deduce that $A.R = B.R = C.R$.

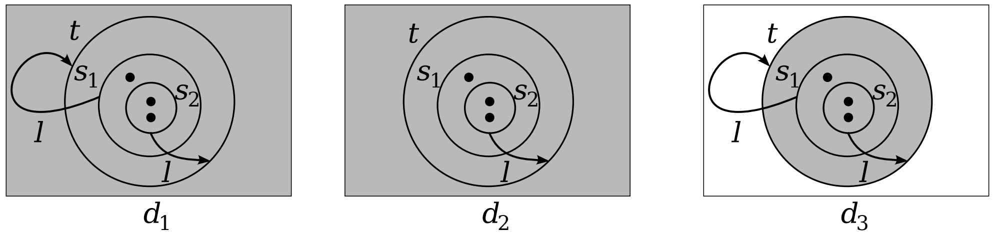


Figure 5.17: Redundancy with no subset relation of source spiders.

There is one case left to consider, in which the source spiders of a redundant arrow a are not a subset of or equal to the source spiders of any set of arrows X that excludes a, where X allows us to infer the redundancy of a. In Figure 5.17, d_1, the arrow (l, s_1, t) is redundant. We can see that this is true if we consider d_2, which illustrates $d_1 - \{(l, s_1, t)\}$. If (U, Ψ', Φ) is an extended interpretation which satisfies d_2, $\Psi'(s_2).\Phi(l) = U$. We know that $\Psi'(s_1) \supseteq \Psi'(s_2)$, and so $\Psi'(s_1).\Phi(l) \supseteq \Psi'(s_2).\Phi(l)$. Thus, $\Psi'(s_1).\Phi(l) = U$. This is only the case because $\Psi'(s_2).\Phi(l) = U$. In other words, there are no elements outside of t in any model for d_1 or d_2. In diagram d_3 the zone outside t is unshaded and so there are models for d_3 in which $U - \Psi'(t) \neq \emptyset$. In this case (l, s_1, t) is not a redundant arrow, as we cannot infer that we can replace it in $d_3 - \{(l, s_1, t)\}$.

Thus, derived contours and the presence of arrows sourced on contours greatly increase the types of syntactic configuration we need to consider when adapting Stapleton's strategy. These two features also interact to result in non-unique minimal and maximal forms. At the underlying level, these issues stem from the fact that generalized constraint diagrams have a second-order semantics. Unlike the completeness proofs for earlier, related notations, we need to reason about extensions rather than interpretations. That is, when we say that a generalized diagram is satisfiable, we posit the existence of an extension that maps spiders and derived contours to subsets of a universe. Rather than showing that two diagrams, D_1 and D_2, have all the same models, we need to show that any valid extension of a model for D_1 is also valid for D_2 and vice versa. While far from being an intractable problem, this gives rise to added complexity. Notwithstanding this difficulty, a completeness proof is an important component of the metatheory of generalized diagrams.

Glossary

τ_d τ_d is the equality relation on the spiders of the generalized unitary diagram d. 35

$[x]_d$ The equivalence class of the spider x under τ_d. 74

$\subseteq$ See sub-diagram. 194

$\subseteq_S$ See syntactic sub-diagram. 167

$\top$ The trivial diagram. 108

$\wedge$-linear normal form A generalized diagram, D, is said to be in $\wedge$-linear normal form if and only if all $\wedge$-labelled nodes in D are linear nodes. 128

β-diagram A generalized unitary diagram is called a β-diagram if it is an α-diagram in which each zone is shaded or contains a spider or both. 56

η A mapping from spiders to zones which returns the zone a spider inhabits. The restriction of the domain of η to the spiders of a given diagram d is denoted η_d. 35

$\perp$ The generalized unitary diagram which represents falsehood. In the abstract syntax, $\perp = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$. 35

γ-diagram A generalized unitary diagram is called a γ-diagram if it is a β-diagram in which each arrow is sourced on a contour. 56

$A(d)$ The set of arrows of the generalized unitary diagram d. 35

$ACC(c, d)$ The add contour components of d for c. 145

$ASTC(x, z, d)$ The add spider with ties components of d for x in z. 148

$ACCD(c, d)$ The add contour component diagrams of d for c. 146

$ASTCD(x, z, d)$ The add spider with ties component diagrams of d for x in z. 149

add contour component Given a generalized unitary diagram, d, and a contour, c, which does not appear in d, then the generalized unitary diagram d' is an add contour component of d for c if d' is obtained by adding c to d so that it splits every zone, given a choice of the habitats for the spiders of d in d' . 145

add contour component diagram A generalized diagram consisting of a single node labelled by a generalized unitary diagram which is an add contour component of a generalized unitary diagram, d, for a contour, c. 146

add spider with ties component Given a generalized unitary diagram, d, a zone, z, in d and a spider, x, which does not appear in d, then the generalized unitary diagram d' is an add spider with ties component of d for z and x if d' is obtained by adding x to z in d so that it is tied to some spider, y, in $S(z, d)$. 148

add spider with ties component diagram A generalized diagram consisting of a single node labelled by a generalized unitary diagram which is an add spider with ties component of a unitary diagram, d, for a spider, x, in a zone, z. 149

$A(l, d)$ The arrows of the generalized unitary diagram, d, which have the label l. 59

$Anc(n, D)$ The ancestors of n in D. 44

arrow A triple of *label*, *source* and *target*, where the label is drawn from $\mathcal{AL}$ while the source and target are arrow ends. 34

arrow end Either a contour or a spider. 34

arrows condition The arrows condition for a generalized unitary diagram, d, asserts that, for each arrow in d, 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. Denoted $AC(d)$. 47

associated contour-source form An associated contour-source form of a generalized unitary diagram, d, is a unitary diagram in contour-source form which has an equivalent meaning to d. 99

associated singleton-τ form An associated singleton-τ form of a generalized unitary diagram, d, is a unitary diagram in singleton-τ form which has an equivalent meaning to d. 97

$C(d)$ The contours of the generalized unitary diagram d. 35

containment condition The containment condition for a generalized unitary diagram, d, asserts that the set represented by a basic region of d is the same as that represented by the containing contour. 48

contour In drawn diagrams, a contour is a closed curve which represents a set. Contours may be given contours, which are labelled to identify the set which is represented, or derived contours, which are unlabelled and represent anonymous subsets. In the abstract syntax, given contours are identified by their labels, whereas derived contours, which have no label, are assumed to represent themselves. 34

contour source form A generalized unitary diagram, d, is said to be in contour source form if and only if each arrow in d is sourced on a contour. 87

$DC(d)$ The set of derived contours of a generalized unitary diagram d. 37

derived contour See contour. 34

$Des(n, D)$ The descendants of n in D. 44

descendants The transitive closure of the immediate descendant relation. 44

disjunctive normal form A generalized diagram is said to be in disjunctive normal form if and only if D is either a linear diagram or has a trivial prefix. 173

existential spider In drawn diagrams, an existential spider is an solid circle which represents the existence of an element in its habitat. All spiders in our system are existential spiders, and they are referred to simply as 'spiders', unless the context requires clarification. 30

extended interpretation An extended interpretation I' of an interpretation I is produced for a given generalized diagram d, such that I' extends I to interpret the spiders and derived contours of d. 46

$form(d)$ The formula for the generalized unitary diagram, d, is the conjunction of the plane tiling, shaded zones, spiders' habitat, spiders distinctness and arrows conditions for d. 47

γ **component** Each unitary α-diagram in single-τ and contour-source forms, d, entails the disjunction of a set of γ-diagrams which we call the γ *components* of d. 101

GCD A generalized constraint diagram. 40

generalized constraint diagram A generalized constraint diagram is a tree whose nodes are labelled by generalized unitary diagrams or by one of the logical connectives $\wedge$ and $\vee$. 40

generalized unitary diagram A generalized unitary constraint diagram, $d = (C, Z, Z^*, S, \eta, \tau, A)$, is a tuple of sets of zones, Z, shaded zones, Z^*, spiders, S, a habitat function, η, which returns the habitat of spiders, an equality relation, τ, on the spiders, and a set of arrows, A. 35

given contour See contour. 34

GuCD A generalized unitary constraint diagram. 35

habitat The habitat of a spider is that zone in which it is placed. 36

$ImmDes(n, D)$ The immediate descendants of n in D. 44

immediate ancestor Given a generalized diagram, D, that contains two nodes, n_1 and n_2, n_1 is the immediate ancestor of n_2 if and only if the edge (n_1, n_2) is in D. 44

immediate descendants Given a generalized diagram, $D = (V, W, E, l)$, that contains a node, n, the immediate descendants of n is the set $\{n' : (n, n') \in E\}$. 44

immediate diagram-labelled ancestor Given a generalized diagram, D, that contains two nodes, n_1 and n_2, n_1 is the immediate diagram-labelled ancestor of n_2 if and only if there is a path of length two from n_1 to n_2 in D. Unlike immediate diagram-labelled descendants, immediate diagram-labelled ancestors are unique. 44

immediate diagram-labelled descendants Given a generalized diagram, D, that contains two nodes, n_1 and n_2, n_2 is an immediate diagram-labelled descendant of n_1 if and only if there is a path of length two from n_1 to n_2 in D. Immediate diagram-labelled descendants are not necessarily unique. 44

inconsistency A generalized unitary diagram, d, is said to be inconsistent or to include inconsistency if, for some arrow label $l \in AL(d)$, there are two sets of arrows X and Y such that $S_t(X, d) - S_t(Y, d) \neq \emptyset$ but $S_s(X, d) - S)s(Y, d) = \emptyset$. 60

induced (sub-diagram) Given a generalized diagram, D_1, with a diagram-labelled node n, we say that the sub-diagram, $D_2 = (V, W, E, l)$, of D_1 with root node n and with $V \cup W = \{n\} \cup Des(n, D_1)$ is the sub-diagram of D_1 induced by n. 109

inhabits A spider is said to inhabit the zone in which it is placed. 36

interpretation An interpretation is a triple (U, Ψ, Φ) where U is a set, Ψ maps contours, regions and zones to subsets of U and Φ maps arrow labels to relations on U. 46

linear connective Given a generalized diagram, D, that contains a node, n, n is a linear connective of D if and only if n is a connective labelled node with an out-degree of one. 43

$MZ(d)$ The set of zones which are missing from the generalized unitary diagram, d. 38

model (compound case) An interpretation, I, is a model for a generalized diagram, D, if I satisfies D. 51

non-linear connective Given a generalized diagram, D, that contains a node, n, n is a non-linear connective of D if and only if n is a connective labelled node with an out-degree of more than one. 43

plane tiling condition The plane tiling condition for a generalized unitary diagram, d, asserts that the union of the sets represented by the zones of d is the universal set. Denoted $PTC(d)$. 47

pushed syntax normal form A generalized diagram, D, is said to be in pushed syntax normal form if and only if for each pair of nodes, n_i and n_j, in D, where n_i is the immediate diagram-labelled ancestor of n_j and the nodes are labelled by the unitary diagrams d_i and d_j respectively, either $d_i \subseteq_S d_j$ or $d_j = \perp$. 167

region A region is a collection of zones. 34

$root(D)$ The function which returns the root node of D. 43

$S(d)$ The set of spiders of a generalized unitary diagram d. 35

$S(z, d)$ The set of spiders inhabiting the zone z in the diagram d. 36

$S(c, d)$ The set of spiders inhabiting the zones which contain the contour c. 37

semantic formula The semantic formula of a generalized (non-unitary) diagram d at vertex v, which is a sentence expressing the logical meaning of d. Denoted $SemForm(v, d)$. 50

shaded zones condition The shaded zones condition for a generalized unitary diagram, d, asserts that all of the elements in the sets represented by shaded zones of d are represented by spiders. Denoted $SZC(d)$. 47

singleton-τ form A generalized unitary diagram, d, is in singleton-τ form if and only if the only ties present in d are those from each spider to itself. 87

source The arrow end at the opposite end of an arrow from the arrow head. 34

source spiders The source spiders of an arrow (l, s, t) are those contained in the source, s. 59

spider See universal spider and existential spider. 30

spiders distinctness condition The spiders distinctness condition for a generalized unitary diagram, d, asserts that distinct spiders represent distinct elements. Denoted $SDC(d)$. 47

spiders' habitat condition The spiders' habitat condition for a generalized unitary diagram, d, asserts that the elements represented by the spiders of d are in the sets represented by their habitats. Denoted $SHC(d)$. 47

S_s $S_s(X, d)$ denotes the union of the source spiders of the arrows contained in the set X, where $X \subseteq A(d)$. 59

$S(s, d)$ The singleton set of spiders containing s. 37

S_t $S_t(X, d)$ denotes the union of the target spiders of the arrows contained in the set X, where $X \subseteq A(d)$. 59

standard extension The standard interpretation of a γ-diagram, d, where Ψ is extended to interpret spiders and derived contours. 67

standard interpretation The standard interpretation of a γ-diagram d is that interpretation in which the universe is the spiders of d and Ψ and Φ are 'natural' mappings from contours, regions and zones to subsets of $S(d)$ and from arrow labels to relations on $S(d)$, respectively. 66

sub-diagram A generalized unitary diagram, d_1, is a sub-diagram of a second unitary diagram, d_2, denoted $d_1 \subseteq d_2$, if and only if the following is true:

1. $d_1 \subseteq_S d_2$,
2. $\eta_{d_1} \subseteq \eta_{d_2}$,
3. each zone which is shaded or missing in d_1 contains the same spiders as in d_2: $\forall z \in Z^*(d_1) \cup MZ(d_1)\,(S(z, d_1) = S(z, d_2))$, and
4. for each pair of spiders $x, y \in S(d_1)$, x and y are joined by a tie in d_1 if and only if they are joined by a tie in d_2: $\forall x, y \in S(d_1)\,((x, y) \in \tau_{d_1} \leftrightarrow (x, y) \in \tau_{d_2})$.

194

sub-diagram (compound) A generalized diagram, $D_1 = (V_1, W_1, E_1, l_1)$, is a sub-diagram of a second generalized diagram, $D_2 = (V_2, W_2, E_2, l_2)$, denoted $D_1 \subseteq D_2$, if and only the following is true:

1. $V_1 \subseteq V_2$,
2. $W_1 \subseteq W_2$,
3. $E_1 = E_2 \cap ((V_1 \times W_1) \cup (W_1 \times V_1))$, and
4. $l_1 = l_2|_{V_1 \cup W_1}$.

41

syntactic sub-diagram A generalized unitary diagram, d_1, is a syntactic sub-diagram of a second unitary diagram, d_2, denoted $d_1 \subseteq_S d_2$, if and only if the following is true:

1. $Z(d_1) \subseteq Z(d_2)$,
2. $Z^*(d_1) \subseteq Z^*(d_2)$,
3. $S(d_1) \subseteq S(d_2)$, and
4. $A(d_1) \subseteq A(d_2)$.

167

target The arrow end at the same end of an arrow as the arrow head. 34

target spiders The target spiders of an arrow (l, s, t) are those contained in the target, t. 59

tie In drawn diagrams, a tie between two spiders is a pair of parallel lines extending between the spiders, and represents equality. In the abstract syntax, two spiders, x and y, in a generalized unitary diagram, d, are tied (i.e. x and y represent the same element) if and only if $(x, y) \in \tau_d$. 34

trivial diagram The generalized unitary diagram which contains no syntax other than the zone $(\emptyset, \emptyset)$. The trivial diagram is true under any interpretation. 108

trivial prefix A generalized diagram, D, is said to have a trivial prefix if the following conditions are true:

1. the root node of D, n_1, is labelled by $\top$,
2. the immediate descendant of n_1 is a non-linear $\vee$-labelled node, and
3. each of the sub-diagrams of D induced by the immediate diagram-labelled descendants of n_1 is a linear sub-diagram of D.

171

universal spider In drawn diagrams, a universal spider is an asterisk which represents universal quantification over the elements in its habitat. Universal spiders are excluded from our system. 30

$VZ(d)$ The Venn zone set of the generalized unitary diagram, d. 38

Venn form (compound) A generalized diagram, D, is said to be in Venn form if and only if all generalized unitary diagrams labelling nodes in D are in Venn form. 140

Venn form (unitary) A generalized unitary diagram, d, is said to be in Venn form if d contains no missing zones. 38

$Z(c, d)$ The zones of the generalized unitary diagram d which are contained in the contour c. 37

zone Given a generalized unitary diagram, d, a zone, (in, out), is a pair of disjoint sets of contour labels of d, where $in \cup out = C(d)$. 34

$Z(d)$ The set of zones of the generalized unitary diagram d. 35

$Z^*(d)$ The set of shaded zones of the generalized unitary diagram d. 35

Bibliography

[1] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development*. Springer, June 2004.

[2] E. Börger, E. Gradel, and E. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[3] J. Burton, G. Stapleton, and A. Hamie. Transforming constraint diagrams. In Philip Cox, Andrew Fish, and John Howse, editors, *Visual Languages and Logic*, pages 62–80, September 2009.

[4] Jim Burton, Gem Stapleton, and Ali Hamie. A decision procedure for a decidable fragment of generalized constraint diagrams. *Journal of Visual Languages & Computing*, November 2010.

[5] P. P. Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, March 1976.

[6] B. H. C. Cheng. A Metamodel-Based Approach to Formalizing UML. *Computer Software and Applications Conference, Annual International*, 0:278+, 2001.

[7] S. Barry Cooper. *Computability Theory (Chapman Hall/CRC Mathematics Series)*. Chapman & Hall/CRC, November 2003.

[8] P. Coppin, J. Burton, and S. Hockema. An Attention Based Theory to Explore Affordances of Textual and Diagrammatic Proofs. In Ashok Goel, Mateja Jamnik, and N. Narayanan, editors, *Diagrammatic Representation and Inference*, volume 6170 of *Lecture Notes in Computer Science*, chapter 27, pages 271–278–278. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[9] Leonardo de Moura and Nikolaj Bjørner. Deciding Effectively Propositional Logic Using DPLL and Substitution Sets. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, chapter 35, pages 410–425. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[10] L. Euler. Lettres a une Princesse d'Allemagne sur divers sujets de physique et de philosophie. *Letters*, 2:102–108, 1775.

[11] N. Fetais and P. Cheng. An Experiment to Evaluate Constraint Diagrams with Novice Users. In Ashok Goel, Mateja Jamnik, and N. Narayanan, editors, *Diagrammatic Representation and Inference*, volume 6170 of *Lecture Notes in Computer Science*, chapter 37, pages 307–309. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[12] A. Fish, J. Flower, and J. Howse. The Semantics of Augmented Constraint Diagrams. *Journal of Visual Languages and Computing*, 16:541–573, 2005.

[13] J. Gil, J. Howse, and S. Kent. Formalising Spider Diagrams. In *Proceedings of IEEE Symposium on Visual Languages (VL99), Tokyo*, pages 130–137. IEEE Computer Society Press, September 1999.

[14] J. Gil, J. Howse, and S. Kent. Towards a formalization of constraint diagrams. In *Proc IEEE Symposia on Human-Centric Computing (HCC '01), Stresa, Italy*, pages 72–79. IEEE Computer Society Press, September 2001.

[15] J. Gil and Y. Sorkin. The Constraint Diagrams Editor. Available at www.cs.technion.ac.il/Labs/ssdl/research/cdeditor/.

[16] C. Gurr and K. Tourlas. Towards the Principled Design of Software Engineering Diagrams. In *Proceedings of 22nd International Conference on Software Engineering*, pages 509–518. ACM Press, 2000.

[17] E. Hammer. *Logic and Visual Information.* CSLI, Stanford, 1995.

[18] D. Harel. On visual formalisms. In J. Glasgow, N. H. Narayan, and B. Chandrasekaran, editors, *Diagrammatic Reasoning*, pages 235–271. MIT Press, 1998.

[19] J. Howse, F. Molina, S. J. Shin, and J. Taylor. Type-syntax and Token-syntax in Diagrammatic Systems. In *Proceedings FOIS-2001: 2nd International Conference on Formal Ontology in Information Systems, Maine, USA*, pages 174–185. ACM Press, 2001.

[20] J. Howse, F. Molina, and J. Taylor. A sound and complete diagrammatic reasoning system. In *Proceedings. ASC 2000: 3rd IASTED International Conference on Artificial Intelligence and Soft Computing*, pages 402–408, Banff, 2000. IASTED/ACTA Press.

[21] J. Howse, F. Molina, and J. Taylor. On the completeness and expressiveness of spider diagram systems. In *Proceedings of 1st International Conference on the Theory and Application of Diagrams*, pages 26–41, Edinburgh, UK, September 2000. Springer.

[22] J. Howse, F. Molina, and J. Taylor. SD2: A sound and complete diagrammatic reasoning system. In *Proceedings VL 2000: IEEE Symposium on Visual Languages, Seattle, USA*, pages 127–136. IEEE Computer Society Press, 2000.

[23] J. Howse, F. Molina, J. Taylor, and S. Kent. Reasoning with Spider Diagrams. In *Proceedings of IEEE Symposium on Visual Languages (VL99), Tokyo*, pages 138–147. IEEE Computer Society Press, September.

[24] J. Howse, F. Molina, J. Taylor, S. Kent, and J. Gil. Spider Diagrams: A Diagrammatic Reasoning System. *Journal of Visual Languages and Computing*, 12(3):299–324, June 2001.

[25] J. Howse and S. Schuman. Precise Visual Modelling. *Journal of Software and Systems Modeling*, 4:310–325, 2005.

[26] J. Howse, G. Stapleton, and Taylor. Spider Diagrams. *LMS Journal of Computation and Mathematics*, 8:145–194, 2005.

[27] S. Kent. Constraint Diagrams: Visualizing Invariants in Object Oriented Modelling. In *Proceedings of OOPSLA97*, pages 327–341. ACM Press, October 1997.

[28] F. Molina. *Reasoning with extended Venn-Peirce diagrammatic systems.* PhD thesis, University of Brighton, 2001.

[29] Omg. OCL 2.0 specification, revision 1.6. Available from http://www.omg.org, 2003.

[30] V. Peckhaus. Leibniz's Influence on 19th Century Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Fall 2009 edition, 2009.

[31] C. S. Peirce. *Collected Papers of Charles Sanders Peirce.* Thoemmes Continuum.

[32] B. Potter, J. Sinclair, and D. Till. *Introduction to Formal Specification and Z (2nd Edition).* Prentice Hall PTR, July 1996.

[33] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A Formalisation of Constraint-Aware Model Transformations. In David Rosenblum and Gabriele Taentzer, editors, *Fundamental Approaches to Software Engineering*, volume 6013 of *Lecture Notes in Computer Science*, chapter 2, pages 13–28. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[34] T. Sheard. Languages of the future. *SIGPLAN Notices*, 39(12):119–132, 2004.

[35] A. Shimojima. Operational constraints in diagrammatic reasoning. In *Logical Reasoning with Diagrams*, pages 27–48. Oxford University Press, 1996.

[36] S. J. Shin. *The Logical Status of Diagrams.* CUP, 1994.

[37] S. J. Shin. *The Iconic Logic of Peirce's Graphs.* Bradford Books, 2002.

[38] G. Stapleton. *Reasoning with Constraint Diagrams.* PhD thesis, University of Brighton, August 2004.

[39] G. Stapleton. A survey of reasoning systems based on Euler diagrams. In *Proceedings of Euler Diagrams 2004, Brighton, UK*, volume 134 of *ENTCS*, pages 127–151, 2005.

[40] G. Stapleton and A. Delaney. Towards Overcoming Deficiencies in Constraint Diagrams. In *Symposium on Visual Languages and Human-Centric Computing*, pages 33–40. IEEE, 2007.

[41] G. Stapleton and A. Delaney. Evaluating and Generalizing Constraint Diagrams. *Journal of Visual Languages and Computing*, 19(4):499–521, 2008.

[42] G. Stapleton, J. Howse, and J. Taylor. A Constraint Diagram Reasoning System. In *Proceedings of International Conference on Visual Languages and Computing*, pages 263–270. Knowledge Systems Insitute, 2003.

[43] G. Stapleton, J. Howse, and J. Taylor. A Decidable Constraint Diagram Reasoning System. *Journal of Logic and Computation*, 15(6):975–1008, December 2005.

[44] G. Stapleton, J. Masthoff, J. Flower, A. Fish, and J. Southern. Automated theorem proving in Euler diagrams systems. *Journal of Automated Reasoning*, 39:431–470, 2007.

[45] G. Stapleton, J. Taylor, J. Howse, and S. Thompson. The Expressiveness of Spider Diagrams Augmented with Constants. *Submitted to Journal of Visual Languages and Computing*, 2005.

[46] G. Stapleton, S. Thompson, J. Howse, and J. Taylor. The Expressiveness of Spider Diagrams. *Journal of Logic and Computation*, 14(6):857–880, December 2004.

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

[48] Unified Modelling Language. The UML website. http://www.uml.org/, 2006.

[49] J. Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *Phil.Mag*, 1880.