

A general framework for sentential and diagrammatic natural deduction

DRAFT

This research was supported by DARPA IPTO

Konstantine Arkoudas and Selmer Bringsjord

July 26, 2005

Abstract

We introduce a unifying framework for heterogeneous natural deduction, combining diagrammatic and sentential reasoning, in the form of a family of denotational proof languages (DPLs). Diagrams are represented as possibly partial descriptions of finite system states. This allows us to deal with incomplete knowledge, which we formalize by admitting sets of values as attributes of system objects. We show how to interpret arbitrary first-order signatures into such system states, and develop a formal semantic framework based on Kleene’s three-valued logic. We extend the assumption-base semantics of DPLs to accommodate diagrammatic reasoning by introducing general inference mechanisms for the valid extraction of information from diagrams and for the incorporation of sentential information into diagrams. A rigorous big-step operational semantics is given, on the basis of which we prove that our framework is sound. We also discuss questions of completeness and efficiency.

1.1 Background

Diagrams have been recognized as valuable representational and reasoning tools since the days of Euclid. Their representational power is often thought to stem from the fact that diagrams have structural correspondences with the objects or situations they represent—they are *analogical representations* in the terminology of Myers and Konolige [29], or *homomorphic representations* in the terminology of Barwise and Etchemendy [6]. In more plain terms, a diagram *resembles* what the diagram depicts, in contrast to sentential descriptions. This was noticed at least as far back as the 19th century, when Charles Peirce observed that a diagram is “naturally analogous to the thing represented” [33].

Consider, for instance, the task of describing some human face. We could perhaps describe the face with a collection of English sentences, or with a set of sentences in some formal language. But such a description is likely to be excessively long and complicated, and hence not particularly informative. A drawing or a picture of the face, on the other hand, will be much more perspicuous, as well as significantly more compact than any sentential representation. Of course, some diagrams are better than others. A talented artist will produce a drawing that is a much more accurate depiction than the scrawlings of a 5-year-old. A digital picture will be even more accurate.¹ So, as Hammer observes [20], being an analogical or homomorphic representation is not a distinguishing feature of diagrams in general, but rather a distinguishing feature of *good* diagrams.

This ability of (good) diagrams is in turn often thought to derive from the fact that diagrams are two-dimensional objects, and therefore spatial relationships in the diagram can directly reflect analogous relationships in the underlying domain (an observation made a while back by Russell [37]). A classic example here is that of maps. We could represent the streets of a city graphically, with a map, or sententially, e.g., by a collection of sentences expressing the various intersections and so forth. The graphical representation is without doubt a more intuitive and effective depiction because its spatial structure is similar to the actual layout of the city; this analogical correspondence is lost in the sentential representation. As another example, consider a map of a lake and try to imagine a sentential description of it. Stenning and Lemon [39] trace this discrepancy to the fact that sentential languages derive from acoustic signals, which are one-dimensional and must therefore rely on a complex syntax for representation, something that is not necessary in the case of diagrams. Nevertheless, it is important to keep in mind that two-dimensionality by itself is neither a necessary nor a sufficient condition for being a diagram [20]. For instance, a representation of a picture by a two-dimensional array of pixels does not classify as a diagram. And by making sufficiently clever conventions, one can

¹In the limiting case, of course, the ultimate representation of an object is the object itself; in that case we have a perfect isomorphism between the representation and the object represented.

very well construct intuitive one-dimensional diagrams. E.g., the following string expresses the fact that the stretch of road between Park Avenue/35th Street and Park Avenue/36th is two-way, whereas that between Park Avenue/36th and Park Avenue/37th is one-way and proceeds from right to left:

Park Ave. + 35th <==> Park Ave. + 36th <== Park Ave. + 37th

Owing to their representational power, diagrams are extensively used in a very wide range of fields. To note just a few examples, witness free-body, energy-level and Feynman diagrams in physics [40], arrow diagrams in algebra and category theory [34], Euler and Venn diagrams in set theory, function graphs in calculus and analysis, planar figures in geometry, bar-, chart- and pie-graphs in economics, circuit, state and timing diagrams in hardware design [24], UML diagrams in software design [36], higraphs in specification [21], visual programming languages [11] and visual logic and specification languages [1, 22, 31], transition graphs in model checking [8], ER-diagrams and hypergraphs in databases [18], semantic networks in AI, graphical user interfaces (GUIs) such as Xerox Parc’s “Magic Lenses” [9], and so on. As the capability of computers to store and manipulate diagrams improves, their use is likely to increase.

Diagrams are not without drawbacks. While they often excel in depicting particular, concrete objects or situations, they are usually not good choices for describing general, abstract structures and relationships. Roughly, the more specific the class of models captured by a diagram, the more successful (and useful) the diagram is likely to be. Spatial constraints tend to pull diagrams toward over-specificity, and end up limiting their generality and expressiveness as a result. To take an extreme example, diagrams cannot express tautological or contradictory information. Expressive limitations can lead to incorrect inferences. It is known that Euler circles [17], for instance, are unsound. This follows from Helly’s theorem in convex topology [14]. A simple illustration of the problem, due to Lemon and Pratt [26], is the following: consider four sets A , B , C , and D , any three of which have non-empty intersections:

$$\begin{aligned} A \cap B \cap C &\neq \emptyset \\ B \cap C \cap D &\neq \emptyset \\ A \cap C \cap D &\neq \emptyset \end{aligned}$$

These are three perfectly consistent premises. But any Euler diagram that tried to depict them graphically would lead to the incorrect conclusion that all four sets have a non-empty intersection (i.e., that $A \cap B \cap C \cap D \neq \emptyset$), which does not in fact follow from the premises. This is a consequence of a special case of Helly’s theorem, which states that if any three out of four convex regions have a non-empty intersection, then all four must have a non-empty intersection. Similar negative results hold for other diagrammatic ways of depicting sets and relationships between them, such as Englebretsen’s linear diagrams [15]; see Lemon’s article [25] for a thorough discussion.

The complexity of diagrammatic reasoning is another issue. Roughly, there are two types of diagrammatic inference. In one of them, exemplified by Euler circles and Venn diagrams, inference is carried out by drawing appropriate diagrams and then reading off the appropriate bits of information from the constructed picture. This type of diagrammatic inference is summarized by the slogan “If you can draw it, it holds.”² In the second type of diagrammatic inference, exemplified in systems such as Hyperproof and in our own $\mathcal{DN}\mathcal{D}\mathcal{L}$, inference is carried out in a more traditional sense,

²For instance, to check the validity of a syllogism with a Venn diagram, all we have to do is draw a figure that represents the premises of the syllogism. When done, the picture itself will tell us whether or not the conclusion follows; nothing further needs to be done in order to verify or falsify the conclusion. Hence, inference in such cases stops with the representation of the premises. In customary reasoning, by contrast, inference only begins *after* the premises have been represented. This is related to the notion of *free rides* [38] in diagrammatic reasoning.

by deriving new diagrams from diagrams that are given as “premises,” or by extracting sentential information from given diagrams. Computational complexity issues have been rigorously investigated for the former, but not for the latter. E.g., for the former, it has been realized that results obtained in studying the complexity of topological inference [19] have a direct bearing on the complexity of drawing diagrams such as Euler circles, and hence on the first type of diagrammatic reasoning. For instance, it has been shown that propositional reasoning with Euler sets is NP-hard, even though reasoning about the same domain can be done polynomially using other representations [25]. In the present work, it will be seen that even though $\mathcal{N}\mathcal{D}\mathcal{L}$ proofs can be checked for soundness in $O(n \log n)$ time in the worst case (where n is the size of the proof), checking $\mathcal{D}\mathcal{N}\mathcal{D}\mathcal{L}$ proofs can take exponential time, although it should be noted that in our case most of the complexity derives from dealing with unknown (incomplete) information. It would appear, therefore, that visual inference, at least in some cases, can be quite significantly more expensive than corresponding sentential reasoning.³

For these and other reasons, researchers have concluded that logical reasoning frameworks must be *heterogeneous* or *hybrid* [6, 30]: they must support both diagrammatic and sentential modes of representation and reasoning, and must allow users to freely combine the two as appropriate. In the attempt to formulate a generic framework for diagrammatic reasoning, one naturally confronts the question of what type of diagrams to use. As Barwise and Etchemendy correctly observe [6], it would be impossible to construct a universal framework for diagrammatic reasoning that relied on a specific type of diagrams. What makes a class of diagrams appropriate for certain problems might make them inappropriate for others. In the example of Barwise and Etchemendy, at different times electrical engineers use state diagrams, circuit diagrams, and timing diagrams to represent and reason about hardware as needed by the appropriate viewpoint at hand (control, logic gates, or timing, respectively). There is no single type of diagram that is uniformly appropriate.

Nevertheless, we observe that much of what we do when we reason with or about diagrams does not depend on how diagrams are drawn or even on what they mean. In this paper we identify what is common in a great variety of instances of diagrammatic reasoning, and proceed to factor it out and extrapolate it into general principles. In the resulting framework, the type of diagrams used may vary from application to application, but the principles by which we reason with and about diagrams remain the same. This is not unlike certain other separations that are familiar from traditional, sentential logic: our vocabulary might vary from application to application (we have different constant, relation, and function symbols as dictated by the problem domain), and the interpretation of the atomic formulas that we can build from that vocabulary will also vary, but the general principles by which we reason with such formulas do not change.

1.2 Introduction

For an arbitrary relation $R \subseteq A_1 \times \dots \times A_n$, we write $D(R)$ for the set $\{A_1, \dots, A_n\}$.

Definition 1.1 *An attribute structure is a pair $\mathcal{A} = (\{A_1, \dots, A_k\}; \mathcal{R})$ consisting of a collection of $k > 0$ sets A_1, \dots, A_k called **attributes**; and a countable collection of computable relations \mathcal{R} such*

³There are alternative viewpoints, however. AI researchers have put forth the notion of *vivid knowledge bases* [16, 27], in which deductive retrieval can be performed particularly efficiently. Such knowledge bases consist only of ground sentences, ground inequalities, and universal quantifications. Etherington et al. [16] claim that “the notion of vivid representations ... corresponds well to the kind of information expressed in pictures” and that “much of the information we gain (i.e., perceptually) occurs naturally in vivid form.” Likewise, Levesque [27] states that “perhaps the main source of vividly represented knowledge is pictorial information.” If that is indeed the case, one would expect pictorial reasoning to be efficient. Lemon [25], however, argues that such claims fail to take into account the type of spatial constraints that limit the expressiveness of diagrammatic representations.

that $D(R) \subseteq \{A_1, \dots, A_k\}$ for each $R \in \mathcal{R}$.

An attribute structure is thus a type of regular heterogeneous algebraic structure [28, 41] (without any operators) whose carriers are called “attributes” for reasons that will become clear in what follows. Note that the relations of an attribute structure are computable owing to the finite size of the attributes. We will tacitly assume that the identity relation on each attribute is included in \mathcal{R} .

We will usually attach a unique *label* l_i to each attribute A_i of a structure \mathcal{A} . Further, when the relations of a structure \mathcal{A} are immaterial, we will identify \mathcal{A} with its attributes. We may then write \mathcal{A} simply as $l_1 : A_1, \dots, l_k : A_k$, where l_i is the label of A_i . Further, we say that \mathcal{A} is *finite* if every attribute of \mathcal{A} is finite.

Definition 1.2 Consider an arbitrary attribute structure \mathcal{A} . An \mathcal{A} -**system** is a pair

$$\mathcal{S} = (\{s_1, \dots, s_n\}; \mathcal{A})$$

consisting of a finite number $n > 0$ of **objects** s_1, \dots, s_n and \mathcal{A} . An attribute of \mathcal{A} may include some (or all) of the objects s_1, \dots, s_n . If that is the case, \mathcal{S} is called **automorphic**.

When \mathcal{A} is obvious from the context or immaterial, we drop references to it and speak simply of “systems” rather than “ \mathcal{A} -systems.”

Example 1.1 Consider a system consisting of a clock c , with two attributes, hours and minutes:

$$(\{c\}; \text{hours} : \{0, \dots, 23\}, \text{minutes} : \{0, \dots, 59\})$$

Another system based on the same attribute structure might consist of two clocks c_1 and c_2 , perhaps indicating New York and Tokyo times, respectively:

$$(\{c_1, c_2\}; \text{hours} : \{0, \dots, 23\}, \text{minutes} : \{0, \dots, 59\})$$

■

Example 1.2 Consider a system comprising the nodes of a three-element linked list, with two attributes, a “data” field consisting of a single ASCII character and a “next” field consisting of another node or the null value:

$$(\{n_1, n_2, n_3\}; \text{data} : \text{Char}, \text{next} : \{n_1, n_2, n_3, \text{null}\})$$

where *Char* is the set of the 256 ASCII symbols and *null* is a special token distinct from $\{n_1, n_2, n_3\}$.

■

Example 1.3 Consider a blocks-world system consisting of three blocks A, B , and C , and a single “position” attribute, where a position is either a block or the floor:

$$(\{A, B, C\}; \text{pos} : \{A, B, C, \text{floor}\})$$

where *floor* is distinct from A, B , and C .

■

Example 1.4 Consider a Hyperproof system consisting of four blocks and three attributes: a pair of integers (i, j) with $0 < i, j < 9$ indicating a grid location; a size (small, medium, or large); and a shape (cube, tetrahedron, or dodecahedron):

$$(\{b_1, b_2, b_3, b_4\}; \text{loc} : \{1, \dots, 8\} \times \{1, \dots, 8\}, \text{size} : \{\text{small}, \text{medium}, \text{large}\}, \text{shape} : \{\text{cube}, \text{tet}, \text{dodec}\})$$

■

For any set S , we write $\mathcal{P}_\infty(S)$ for the set of all finite subsets of S .

Definition 1.3 A state of a system $\mathcal{S} = (\{s_1, \dots, s_n\}, \{A_1, \dots, A_k\})$ is a set of functions $\mathcal{D}_\mathcal{S} = \{a_1, \dots, a_k\}$ where each a_i is a map from $\{s_1, \dots, s_n\}$ to the non-empty set of all finite subsets of A_i , i.e.,

$$a_i : \{s_1, \dots, s_n\} \rightarrow \mathcal{P}_\infty(A_i) \setminus \emptyset$$

We refer to each a_i as the **ascription** into A_i . An ascription a_i is a **valuation** if it maps every object to a singleton, i.e., if $|a_i(s_j)| = 1$ for every $j = 1, \dots, n$. We may thus view a valuation as mapping every object to a unique attribute value. A **world** is a state in which every ascription is a valuation.

A system that is based on a finite attribute structure has

$$2^{(|A_i|-1)^n}{}^k$$

states, where n is the number of objects and k the number of attributes.⁴ To simplify notation, when a is a valuation that maps an object s to a singleton $\{v\}$, we write $a(s) = v$ instead of $a(s) = \{v\}$. Further, we will often use the label l_i of an attribute A_i to denote the ascription into A_i . That is, we are overloading the label symbols: sometimes l_i will just stand for the attribute A_i and sometimes for a_i , the (unique) ascription into A_i ; the context will always eliminate any ambiguity. As an additional convention, let

$$\begin{aligned} \sigma &= \{a_1, \dots, a_k\} \\ \sigma' &= \{a'_1, \dots, a'_k\} \\ \sigma'' &= \{a''_1, \dots, a''_k\} \\ &\vdots \end{aligned}$$

be states of some system $\mathcal{S} = (\{s_1, \dots, s_n\}; l_1 : A_1, \dots, l_k : A_k)$. We then write l_i, l'_i, l''_i, \dots for the ascriptions a_i, a'_i, a''_i, \dots (for any $i \in \{1, \dots, k\}$). Likewise, if $\sigma_1 = \{a_1^1, \dots, a_k^1\}, \dots, \sigma_m = \{a_1^m, \dots, a_k^m\}$, we write l_1^i, \dots, l_k^i for the ascriptions a_1^i, \dots, a_k^i , respectively, for any $i \in \{1, \dots, m\}$.

Our notion of systems and states is similar to the corresponding notions in the model checking field [12, 8], where a system is represented by a collection of variables and a state of a system is modeled by an assignment of a value (drawn from an appropriate domain) to each variable.

Example 1.5 Consider the single-clock system of Example 1.1:

$$(\{c\}; \text{hours} : \{0, \dots, 23\}, \text{minutes} : \{0, \dots, 59\})$$

⁴Note that our term “system state” corresponds roughly to what Barwise et al. [7] refer to as “situation.” Our notion is much more general, as will be seen.

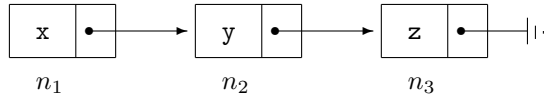


Figure 1.1: A linked list world.

A complete state of this system is given by the following two valuations:

$$hours(c) = 15, minutes(c) = 47$$

indicating a time of 3:47 PM. This is a particular “world” of our clock system.

Suppose we know that it is between 2:30 and 3 past midnight, but we do not know exactly how many minutes past 2:30 it is. This state of knowledge can be captured by the following partial state:

$$hours(c) = 2, minutes(c) = \{31, \dots, 59\}$$

Complete lack of information about the time of the system is indicated by the state:

$$hours = \{0, \dots, 23\}, minutes(c) = \{0, \dots, 59\}$$

■

Example 1.6 Consider the three-element linked-list system of Example 1.2.

The state

$$data(n_1) = x, data(n_2) = y, data(n_3) = z, next(n_1) = n_2, next(n_2) = n_3, next(n_3) = null$$

depicts the world shown in Figure 1.1. The state

$$data(n_1) = Char, data(n_2) = \{a, b\}, data(n_3) = z, next(n_1) = n_2, next(n_2) = \{n_1, n_3\}, next(n_3) = null$$

depicts a system in which we do not know the data field of the first node, we know that the data field of the second node is either *a* or *b*, we know that the next field of the second node is either n_2 or n_3 , and we have fixed values for the remaining node attributes.

■

Example 1.7 Consider the blocks world system of Example 1.3. The state

$$pos(A) = B, pos(B) = floor, pos(C) = floor$$

depicts the blocks world shown in Figure 1.2. The state

$$pos(A) = \{A, B, C, floor\}, pos(B) = \{A, B, C, floor\}, pos(C) = \{A, B, C, floor\}$$

signifies complete lack of information about the positions of the blocks.

■

Example 1.8 Consider the Hyperproof system of Example 1.4. The state

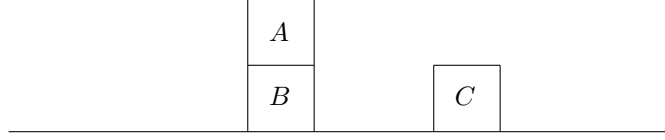


Figure 1.2: A blocks world.

$loc(b_1)$	$=$	$(1, 1)$	$size(b_1)$	$=$	$\{small, medium\}$	$shape(b_1)$	$=$	$cube$
$loc(b_2)$	$=$	$(5, 3)$	$size(b_2)$	$=$	$small$	$shape(b_2)$	$=$	tet
$loc(b_3)$	$=$	$(2, 6)$	$size(b_3)$	$=$	$large$	$shape(b_3)$	$=$	$\{tet, dodec\}$
$loc(b_4)$	$=$	$\{(7, 1), (7, 2), \dots, (7, 8)\}$	$size(b_4)$	$=$	$medium$	$shape(b_4)$	$=$	$dodec$

should be self-explanatory at this point. ■

We may think of partial states as mental models of situations, representing various states of knowledge ranging from completely specific to completely general.

Consider a system $\mathcal{S} = (\{s_1, \dots, s_n\}; l_1 : A_1, \dots, l_k : A_k)$. We say that a state σ of \mathcal{S} is *at least as general* as a state σ' , written $\sigma \sqsupseteq \sigma'$, iff $l_i(s_j) \supseteq l'_i(s_j)$ for every $i = 1, \dots, k$ and $j = 1, \dots, n$. Conversely, we say that σ' is *at least as specific as* σ , written $\sigma' \sqsubseteq \sigma$. Finally, σ is more general than σ' (or σ' is more specific than σ), written $\sigma \sqsubset \sigma'$ (respectively, $\sigma' \sqsubset \sigma$), iff $\sigma \sqsupseteq \sigma'$ and $\sigma \neq \sigma'$ (respectively, if $\sigma' \sqsubseteq \sigma$ and $\sigma \neq \sigma'$). If $\sigma' \sqsubset \sigma$, we say that σ' is an **extension** of σ .⁵ Note that worlds do not have any extensions.

Consider, for instance, the system of Example 1.1:

$$(\{c_1, c_2\}; hours : \{0, \dots, 23\}, minutes : \{0, \dots, 59\})$$

The state

$$\begin{aligned} hours'(c_1) &= \{13, 14\} \\ minutes'(c_1) &= \{55\} \\ hours'(c_2) &= \{6, 7\} \\ minutes'(c_2) &= \{9, 10\} \end{aligned} \tag{1.1}$$

is an extension of the state

$$\begin{aligned} hours(c_1) &= \{13, 14, 15\} \\ minutes(c_1) &= \{55\} \\ hours(c_2) &= \{6, 7\} \\ minutes(c_2) &= \{9, 10, 11\} \end{aligned} \tag{1.2}$$

The set of all states of \mathcal{S} is arranged into a rich partial order corresponding to the join (union) semi-lattice

$$(\mathcal{P}_\infty(A_1) \setminus \emptyset) \times \dots \times (\mathcal{P}_\infty(A_k) \setminus \emptyset)$$

under set union.

⁵The terminology sounds somewhat paradoxical, since an extension of a state is one that assigns *fewer* attribute values to each system object, thereby making our knowledge of the system more specific. This is similar to the terminology of object-oriented class hierarchies, where we say that “human” is an extension of “mammal” to mean that the former is in fact a subset of the latter.

1.3 Interpreting first-order languages into system states

Consider a first-order vocabulary $\Sigma = (\mathcal{C}; \mathfrak{R}; \mathfrak{V})$ consisting of a set of constant symbols \mathcal{C} ; a set of relation symbols \mathfrak{R} , with each $R \in \mathfrak{R}$ having a unique positive arity; and a set of variables \mathfrak{V} . An **interpretation** of Σ into an attribute structure $\mathcal{A} = (\{l_1 : A_1, \dots, l_k : A_k\}; \mathcal{R})$ is a mapping I that assigns, to each relation symbol $R \in \mathfrak{R}$ of arity n :

1. a relation $R^I \in \mathcal{R}$ of some arity m , called the **realization** of R :

$$R^I \subset A_{i_1} \times \dots \times A_{i_m}$$

(where we might have $m \neq n$) and

2. a list of m pairs

$$[(l_{i_1}, j_1), \dots, (l_{i_m}, j_m)]$$

called the **profile** of R and denoted by $Prof(R)$, with $1 \leq j_x \leq n$ for each $x = 1, \dots, m$.

In what follows, fix a signature $\Sigma = (\mathcal{C}; \mathfrak{R}; \mathfrak{V})$, an attribute structure

$$\mathcal{A} = (\{l_1 : A_1, \dots, l_k : A_k\}; \mathcal{R})$$

and an interpretation I of Σ into \mathcal{A} .

Suppose now that we are given an \mathcal{A} -system $\mathcal{S} = (\{s_1, \dots, s_n\}; \mathcal{A})$. We define a **constant assignment** as a partial function ρ from \mathcal{C} to $\{s_1, \dots, s_n\}$; while a **variable assignment** is a total function χ from \mathfrak{V} to $\{s_1, \dots, s_n\}$. We write $Dom(\rho)$ for the domain of a constant assignment ρ , i.e., the set of all and only those constant symbols for which ρ is defined.

Formulas F over Σ are defined as usual, with a term t being either a variable or a constant symbol. Notions such as free variable occurrences, alphabetic equivalence, etc., are defined in the standard way. We will regard alphabetic equivalent formulas as identical. A sentence is a formula without any free variable occurrences. We define $t^{\rho, \chi}$ as $\rho(c)$ if t is a constant symbol c and as $\chi(v)$ if t is a variable v . Since ρ is a partial function, $t^{\rho, \chi}$ may be undefined.

By a *named state* we will mean a pair $(\sigma; \rho)$ consisting of a state σ and a constant assignment ρ . We say that a named state $(\sigma'; \rho')$ is an extension of a named state $(\sigma; \rho)$ iff σ' is an extension of σ (i.e., $\sigma' \sqsupset \sigma$) and ρ' is an extension of ρ .⁶

We will now show how to assign a truth value to any formula F , given a named state $(\sigma; \rho)$ (of an \mathcal{A} -system $\mathcal{S} = (\{s_1, \dots, s_n\}; \mathcal{A})$) along with a variable assignment χ . This is done by formally defining a mapping $I_{(\sigma; \rho), \chi}$ from the set of all formulas to the three-element set

$$\{\mathbf{true}, \mathbf{false}, \mathbf{unknown}\}$$

as follows.

First consider an atomic formula $R(t_1, \dots, t_n)$, where R is a relation symbol of arity n and profile $[(l_{i_1}, j_1), \dots, (l_{i_m}, j_m)]$. We set

$$I_{(\sigma; \rho), \chi}(R(t_1, \dots, t_n)) = \begin{cases} \mathbf{true} & \text{if } \forall \alpha_1 \in l_{i_1}(t_{j_1}^{\rho, \chi}) \dots \forall \alpha_m \in l_{i_m}(t_{j_m}^{\rho, \chi}) . R^I(\alpha_1, \dots, \alpha_m) \\ \mathbf{false} & \text{if } \forall \alpha_1 \in l_{i_1}(t_{j_1}^{\rho, \chi}) \dots \forall \alpha_m \in l_{i_m}(t_{j_m}^{\rho, \chi}) . \neg R^I(\alpha_1, \dots, \alpha_m) \\ \mathbf{unknown} & \text{otherwise} \end{cases} \quad (1.3)$$

⁶A partial function f' is an extension of a partial function f if $f'(x) = f(x)$ for all x in the domain of f .

In the first two cases above we tacitly assume—in the interest of readability—that $t_{j_x}^{\rho, \chi}$ is defined for every $x = 1, \dots, m$. If not, then the value of $I_{(\sigma; \rho), \chi}(R(t_1, \dots, t_n))$ is **unknown**.

Sentential combinations of formulas are interpreted according to the classical three-valued Kleene scheme. For instance:

$$I_{(\sigma; \rho), \chi}(F_1 \wedge F_2) \begin{cases} \mathbf{true} & \text{if } I_{(\sigma; \rho), \chi}(F_1) = \mathbf{true} \text{ and } I_{(\sigma; \rho), \chi}(F_2) = \mathbf{true} \\ \mathbf{false} & \text{if } I_{(\sigma; \rho), \chi}(F_1) = \mathbf{false} \text{ or } I_{(\sigma; \rho), \chi}(F_2) = \mathbf{false} \\ \mathbf{unknown} & \text{otherwise} \end{cases} \quad (1.4)$$

Finally, quantified formulas are evaluated as follows:

$$I_{(\sigma; \rho), \chi}(\forall v . F) \begin{cases} \mathbf{true} & \text{if } I_{(\sigma; \rho), \chi[v \mapsto s_i]}(F) = \mathbf{true} \text{ for every } i = 1, \dots, n \\ \mathbf{false} & \text{if } I_{(\sigma; \rho), \chi[v \mapsto s_i]}(F) = \mathbf{false} \text{ for some } i \in \{1, \dots, n\} \\ \mathbf{unknown} & \text{otherwise} \end{cases} \quad (1.5)$$

and

$$I_{(\sigma; \rho), \chi}(\exists v . F) \begin{cases} \mathbf{true} & \text{if } I_{(\sigma; \rho), \chi[v \mapsto s_i]}(F) = \mathbf{true} \text{ for some } i \in \{1, \dots, n\} \\ \mathbf{false} & \text{if } I_{(\sigma; \rho), \chi[v \mapsto s_i]}(F) = \mathbf{false} \text{ for every } i = 1, \dots, n \\ \mathbf{unknown} & \text{otherwise} \end{cases} \quad (1.6)$$

where $\chi[v \mapsto s_i]$ is the variable assignment that maps v to s_i and every other variable v' to $\chi(v')$.

The following result is proved by a straightforward induction on the structure of F . This is the three-valued-logic version of the standard “coincidence theorem” of universal algebra and logic, which states that two variable assignments that agree on the free variables of a formula F are indistinguishable for the purposes of determining the truth value of F .

Lemma 1.1 *If $\chi_1(v) = \chi_2(v)$ for every variable v that has a free occurrence in F , then*

$$I_{(\sigma; \rho), \chi_1}(F) = I_{(\sigma; \rho), \chi_2}(F)$$

Example 1.9 Consider the signature $\Sigma_1 = (\mathfrak{C}_{clock}; \mathfrak{R}_{clock}; \mathfrak{V}_{clock})$ where the set of constant symbols is

$$\mathfrak{C}_{clock} = \{\mathbf{c}_1, \mathbf{c}_2, \dots\}$$

the set of variables is $\mathfrak{V}_{clock} = \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ and the set of relation symbols is

$$\mathfrak{R}_{clock} = \{\mathbf{PM}, \mathbf{AM}, \mathbf{Ahead}, \mathbf{Behind}\}$$

with **PM**, **AM** unary and **Ahead**, **Behind** binary.

Consider now the attribute structure

$$Clock = (\mathit{hours} : \{0, \dots, 23\}, \mathit{minutes} : \{0, \dots, 59\}; \{R_1, R_2, R_3, R_4\})$$

where $R_1 \subseteq \mathit{hours}$, $R_2 \subseteq \mathit{hours}$,

$$R_3 \subseteq \mathit{hours} \times \mathit{minutes} \times \mathit{hours} \times \mathit{minutes}$$

$$R_4 \subseteq \mathit{hours} \times \mathit{minutes} \times \mathit{hours} \times \mathit{minutes}$$

defined as follows: $R_1(h) \Leftrightarrow h > 11$, $R_2(h) \Leftrightarrow h \leq 11$,

$$R_3(h_1, m_1, h_2, m_2) \Leftrightarrow h_1 > h_2 \vee (h_1 = h_2 \wedge m_1 > m_2)$$

and

$$R_4(h_1, m_1, h_2, m_2) \Leftrightarrow h_1 \leq h_2 \vee (h_1 = h_2 \wedge m_1 \leq m_2)$$

We define an interpretation I of Σ_1 into this attribute structure by specifying a unique relation (in the structure) and a unique profile for each symbol in \mathfrak{R}_{clock} . In particular, we set $\text{PM}^I = R_1$, $\text{AM}^I = R_2$, $\text{Ahead}^I = R_3$, $\text{Behind}^I = R_4$ and:

$$\begin{aligned} \text{Prof}(\text{PM}) &= [(hours, 1)] \\ \text{Prof}(\text{AM}) &= [(hours, 1)] \\ \text{Prof}(\text{Ahead}) &= [(hours, 1), (minutes, 1), (hours, 2), (minutes, 2)] \\ \text{Prof}(\text{Behind}) &= [(hours, 1), (minutes, 1), (hours, 2), (minutes, 2)] \end{aligned}$$

■

Example 1.10 Consider the system $(\{c_1, c_2\}; \text{Clock})$, where Clock is the attribute structure introduced in Example 1.9. Let σ be the following state of this system:

$$\begin{aligned} hours(c_1) &= \{9, 13\} \\ minutes(c_1) &= 12 \\ hours(c_2) &= 8 \\ minutes(c_2) &= 27 \end{aligned}$$

and let ρ be the partial constant assignment that maps c_1 to c_1 and c_2 to c_2 . We claim that the sentence $\text{Ahead}(c_1, c_2)$ is true in $(\sigma; \rho)$, for any variable assignment χ . Indeed, consider an arbitrary χ . Recalling the profile of Ahead , definition (1.3) tells us that in order to have

$$I_{(\sigma; \rho), \chi}(\text{Ahead}(c_1, c_2)) = \mathbf{true}$$

we must have $R(v_1, v_2, v_3, v_4)$ for all

$$\begin{aligned} v_1 &\in hours(c_1^{\rho, \chi} = c_1) = \{9, 13\} \\ v_2 &\in minutes(c_1^{\rho, \chi} = c_1) = \{12\} \\ v_3 &\in hours(c_2^{\rho, \chi} = c_2) = \{8\} \\ v_4 &\in minutes(c_2^{\rho, \chi} = c_2) = \{27\} \end{aligned}$$

i.e., we must have $R_3(9, 12, 8, 27)$ and $R_3(13, 12, 8, 27)$. Both of these hold according to the definition of R_3 , since $9 > 8$ and $13 > 8$.

As another example, the sentence $\text{PM}(c_1) \wedge \neg \text{PM}(c_1)$ evaluates to **unknown** in $(\sigma; \rho)$, despite being patently inconsistent, because, intuitively, in $(\sigma; \rho)$ we do not know whether c_1 is prior to midnight or after it (one possibility is 9, which is AM, and the other is 13, which is PM). Accordingly, $\text{PM}(c_1)$ evaluates to **unknown**, hence $\neg \text{PM}(c_1)$ also evaluates to **unknown**, and therefore their conjunction evaluates to **unknown** as well. It is instructive to see why, precisely, $\text{PM}(c_1)$ evaluates to **unknown**. Recall that the interpretation of PM is the unary relation R_1 , which holds of a given hour h iff $h > 11$; and that the profile of PM is $[(hours, 1)]$. Accordingly, for $\text{PM}(c_1)$ to be true in $(\sigma; \rho)$ (and arbitrary χ), we must have $R_1(v_1)$ for all

$$v_1 \in hours(c_1^{\rho, \chi} = c_1) = \{9, 13\}$$

i.e., we must have $9 > 11$ and $13 > 11$, which is clearly false. Likewise, for $\text{PM}(c_1)$ to come out **false** in $(\sigma; \rho) \chi$, we must have $\neg R_1(9)$ and $\neg R_1(13)$, which is also false. Accordingly,

$$I_{(\sigma; \rho), \chi}(\text{PM}(c_1)) = \mathbf{unknown}$$

by (1.3). ■

The following is a direct consequence of the finite size of the ascription values:

Lemma 1.2 *The function $I_{(\sigma; \rho), \chi}$ is computable for any named state $(\sigma; \rho)$ and computable variable assignment χ .*

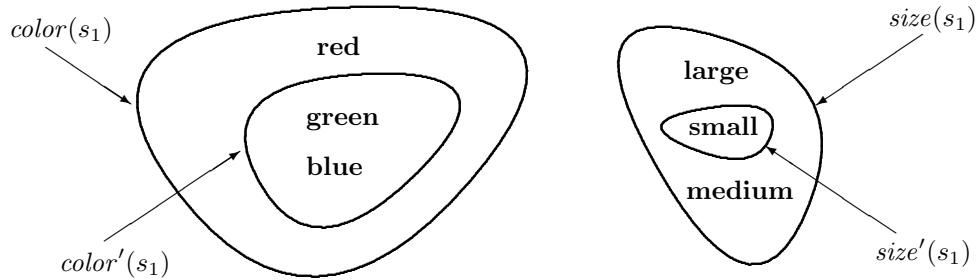
Accordingly, we can effectively compute the truth value of any sentence in any named state.

Next we formalize the important notion of alternative extensions and state entailment.

Definition 1.4 *Let σ' be an extension of a state σ . We say that a state σ'' is an **alternative extension** of σ modulo σ' , written $\text{Alt}(\sigma, \sigma', \sigma'')$, iff there is an attribute l_i and object s_j such that:*

1. $l'_i(s_j) \subset l_i(s_j)$;
2. $l''_i(s_j) = l_i(s_j) \setminus l'_i(s_j)$; and
3. for all attributes l_{i_1} and objects s_{j_1} , if $i_1 \neq i$ and $j_1 \neq j$ then $l''_{i_1}(s_{j_1}) = l_{i_1}(s_{j_1})$.

As a simple example, consider a system consisting of one object s_1 with two attributes, color and size, and suppose that σ stipulates **red**, **green**, and **blue** as the possible color values of s_1 and **large**, **medium** and **small** as its possible size values; and suppose that σ' extends σ by limiting the color values of s_1 to **green** and **blue** and its size to **small**:



What counts as an alternative extension of σ (modulo σ')? Considering that σ' essentially states that the color of s_1 is either green or blue and that its size is small, we could differ from it in one of the following respects:

<i>Color of s_1</i>	<i>Size of s_1</i>
{red}	{large, medium}
{red}	{small}
{green, blue}	{large, medium}

That is, we could either choose to (1) disagree with the color, and either disagree or agree with the size (the latter choice is immaterial in light of the first disagreement), resulting in the top two rows of the table above, or (2) agree with the color but disagree with the size, which leads to the third row. Given that set membership indicates disjunctive information, we can collapse the first two possibilities, obtaining:

<i>Color of s_1</i>	<i>Size of s_1</i>
{red}	{small, large, medium}
{green, blue}	{large, medium}

These are the only two alternative extensions of σ modulo σ' . In general, given an arbitrary extension $\sigma' \sqsubset \sigma$, we can effectively construct all the alternative extensions of σ modulo σ' . There are m such extensions, where m is the number of pairs $(i; j)$ such that $l'_i(s_j) \neq l_i(s_j)$, or equivalently, such that $l'_i(s_j) \subset l_i(s_j)$; i.e., the number of pairs of attributes and objects whose corresponding ascription values changed in going from σ to σ' . We can generate these states by taking the complement of the ascription value of each such pair in σ' ⁷ (clause 2 of Definition 1.4) while reverting the other $m - 1$ pairs to their σ values (clause 3 of Definition 1.4).

It is noteworthy that in determining the alternative extensions of σ modulo σ' , we only consider those objects and those attributes that are changed by σ' . We ignore those ascription assignments that remain the same in going from σ to σ' . As another example, there are two states that are alternative extensions of (1.2) modulo (1.1). In one of them we keep the same *hours* values of c_1 ($\{13, 14\}$) but complement the *minutes* value of c_2 to obtain $\{11\}$; while in the other alternative we keep the original *minutes* value of c_2 ($\{9, 10, 11\}$) but complement the *hours* value of c_1 to obtain $\{15\}$. However, in both cases the *minutes* of c_1 and *hours* of c_2 remain the same as they were in the original state (1.2), because neither of them was modified by (1.1).

The following lemma has a straightforward proof:

Lemma 1.3 *The relation Alt is symmetric, in that a state σ_2 is an extension of σ modulo σ_1 iff σ_1 is an extension of σ modulo σ_2 . In symbols, $Alt(\sigma, \sigma_1, \sigma_2)$ iff $Alt(\sigma, \sigma_2, \sigma_1)$.*

We can formulate a more general notion of an alternative extension, modulo a number of states, by defining the union of several extensions:

Definition 1.5 *Let $\sigma_1, \dots, \sigma_n$ be extensions of a state σ . We define the **union** of those states, denoted $\cup(\sigma_1, \dots, \sigma_n)$, as the unique state σ' such that for every attribute l_i and object s_j ,*

$$l'_i(s_j) = \bigcup_{x=1}^n l_i^x(s_j)$$

Definition 1.6 *Let $\sigma_1, \dots, \sigma_m$ be m extensions of a state σ , $m \geq 1$. We say that σ' is an alternative extension of σ modulo $\sigma_1, \dots, \sigma_m$, written*

$$Alt(\sigma, \{\sigma_1, \dots, \sigma_m\}, \sigma'),$$

iff σ' is an alternative extension of σ modulo $\cup(\sigma_1, \dots, \sigma_m)$.

We can extend this general notion of alternative extensions to named states as follows:

⁷The complement with respect to the corresponding ascription value in σ .

Definition 1.7 Let $(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m), (\sigma'; \rho')$ be $m+1$ extensions of a named state $(\sigma; \rho)$, $m > 0$. We say that $(\sigma'; \rho')$ is **an alternative extension** of $(\sigma; \rho)$ modulo $(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)$, written $\text{Alt}((\sigma; \rho), \{(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)\}, (\sigma'; \rho'))$, iff $\text{Dom}(\rho') = \text{Dom}(\rho_1) \cup \dots \cup \text{Dom}(\rho_m)$, and one of the following two conditions obtains:

1. $\text{Alt}(\sigma, \{\sigma_1, \dots, \sigma_m\}, \sigma')$; or
2. there is some constant symbol $c \in \text{Dom}(\rho')$ such that for every $i = 1, \dots, m$, if $c \in \text{Dom}(\rho_i)$ then $\rho'(c) \neq \rho_i(c)$.

We can mechanically generate all alternative extensions of $(\sigma; \rho)$ modulo $(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)$ by first generating all alternative extensions of σ modulo $\sigma_1, \dots, \sigma_m$ and then combining those with different constant assignments in accordance with the above definition.

Finally, we introduce the following notion of state entailment:

Definition 1.8 Let $(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)$ be extensions of a named state $(\sigma; \rho)$, and let β be a finite set of formulas. We say that $(\sigma; \rho)$ **entails** $(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)$ **with respect to** β , written $(\sigma; \rho) \models_{\beta} \{(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)\}$, iff for every $(\sigma'; \rho')$ such that

$$\text{Alt}((\sigma; \rho), \{(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)\}, (\sigma'; \rho'))$$

there is some $F \in \beta$ such that for all χ

$$I_{(\sigma'; \rho').\chi}(F) = \mathbf{false}$$

When $n = 1$ we drop the braces and write $(\sigma; \rho) \models_{\beta} (\sigma_1; \rho_1)$ instead of $(\sigma; \rho) \models_{\beta} \{(\sigma_1; \rho_1)\}$. This captures the intuition that any world that extends the state $(\sigma; \rho)$ and satisfies the formulas in β must also extend one of the states $(\sigma'; \rho')$, in the sense that any alternative way of extending $(\sigma; \rho)$ will end up falsifying some element of β . (Of course, if there are no alternative ways of extending $(\sigma; \rho)$ then the entailment holds vacuously, even if $\beta = \emptyset$.) Determining whether or not $(\sigma; \rho) \models_{\beta} \{(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)\}$ is decidable, since we can mechanically enumerate all the alternative extensions of $(\sigma; \rho)$ modulo $(\sigma_1; \rho_1), \dots, (\sigma_m; \rho_m)$.

1.4 A family of diagrammatic natural deduction languages

We now introduce $\mathcal{DN}\mathcal{DL}$, a family of natural deduction languages in the DPL tradition [2] that combines sentential and diagrammatic reasoning. A concrete instance of $\mathcal{DN}\mathcal{DL}$ is obtained by fixing a vocabulary $\Sigma = (\mathcal{C}; \mathfrak{R}; \mathfrak{V})$, an attribute structure $\mathcal{A} = (\{l_1 : A_1, \dots, l_k : A_k\}; \mathcal{R})$, and an interpretation I of \mathfrak{R} into \mathcal{A} . We assume in the sequel that Σ , \mathcal{A} , and I have been given. The terms and formulas of the language are defined as described in Section 1.3. We write $F[v \mapsto t]$ to denote the formula obtained by replacing every free occurrence of v by the term t (taking care to rename F if necessary to avoid variable capture).

1.4.1 Abstract syntax

The abstract syntax of proofs is defined by the grammars below. There are two syntactic categories of proofs, sentential and diagrammatic. We use the letters D and Δ to range over the two, respectively.

The bold-faced symbol **D** will range over the union of sentential and diagrammatic deductions.

$$\begin{aligned}
D & ::= \textit{RuleApp} \mid \mathbf{assume} \ F \ D \mid F \ \mathbf{by} \ D \\
& \mid D_1; D_2 \\
& \mid \Delta; D \\
& \mid \mathbf{pick-any} \ x \ D \\
& \mid \mathbf{pick-witness} \ w \ \mathbf{for} \ \exists x . F \ D \\
& \mid \mathbf{specialize} \ \forall x . F \ \mathbf{with} \ t \\
& \mid \mathbf{ex-generalize} \ \exists x . F \ \mathbf{from} \ t \\
& \mid \mathbf{cases from} \ F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow D_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow D_n \\
& \mid \mathbf{observe} \ F \\
\\
\Delta & ::= D; \Delta \\
& \mid \Delta_1; \Delta_2 \\
& \mid \mathbf{claim} \ (\sigma; \rho) \\
& \mid (\sigma; \rho) \ \mathbf{by thinning with} \ F_1, \dots, F_n \\
& \mid (\sigma; \rho) \ \mathbf{by widening} \\
& \mid (\sigma; \rho) \ \mathbf{by absurdity} \\
& \mid \mathbf{cases from} \ F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow \Delta_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow \Delta_n \\
& \mid \mathbf{cases} \ F_1 \vee F_2: F_1 \rightarrow \Delta_1 \mid F_2 \rightarrow \Delta_2 \\
& \mid \mathbf{pick-witness} \ w \ \mathbf{for} \ \exists x . F \ \Delta \\
\\
\mathbf{D} & ::= D \mid \Delta
\end{aligned}$$

where the syntax of inference *rule applications* is as follows:

$$\begin{aligned}
\textit{RuleApp} & ::= \mathbf{claim} \ F \\
& \mid \mathbf{modus-ponens} \ F \Rightarrow G, F \\
& \mid \mathbf{modus-tollens} \ F \Rightarrow G, \neg G \\
& \mid \mathbf{double-negation} \ \neg \neg F \\
& \mid \mathbf{absurd} \ F, \neg F \\
& \mid \mathbf{left-and} \ F \wedge G \\
& \mid \mathbf{right-and} \ F \wedge G \\
& \mid \mathbf{both} \ F, G \\
& \mid \mathbf{left-either} \ F, G \\
& \mid \mathbf{right-either} \ F, G \\
& \mid \mathbf{cases} \ F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G \\
& \mid \mathbf{left-iff} \ F \Leftrightarrow G \\
& \mid \mathbf{right-iff} \ F \Leftrightarrow G \\
& \mid \mathbf{equiv} \ F \Rightarrow G, G \Rightarrow F
\end{aligned}$$

Note that while **cases** is a primitive inference rule for sentential deductions, it is a native syntax form in the case of diagrammatic deductions. The reason for this is that, in the case of a sentential case analysis of a disjunction $F_1 \vee F_2$, we can separately establish the required conditionals $F_1 \Rightarrow G$ and $F_2 \Rightarrow G$ using the **assume** form, and then apply **cases** to $F_1 \vee F_2$, $F_1 \Rightarrow G$ and $F_2 \Rightarrow G$ to obtain G . But there is no analogue to **assume** for diagrammatic deductions (there need not be) and therefore the

hypothetical reasoning of assuming F_1 or F_2 and then proceeding to establish a common system state (rule [C2]) must occur “in-place”. Note that since this syntax is abstract, it makes no commitments on the particular representation of system states. They could be concretely represented either as diagrams or as expressions in a certain language, e.g. expressions of the form

$$\mathbf{shape}(b_1) = \mathbf{cube}, \mathbf{size}(b_1) = \{\mathbf{small}, \mathbf{medium}\}, \dots$$

1.4.2 Evaluation semantics

An **assumption base** β is simply a finite set of formulas. A **knowledge base** γ is defined as a pair $(\beta; (\sigma; \rho))$ consisting of an assumption base β and a named state $(\sigma; \rho)$. Our formal semantics is given by axioms and rules that establish judgments of the form

$$\gamma \vdash \mathbf{D} \rightsquigarrow F$$

and

$$\gamma \vdash \mathbf{D} \rightsquigarrow (\sigma; \rho)$$

which should be read as “In the context of γ , deduction \mathbf{D} derives F (or $(\sigma; \rho)$).”

The semantics of most sentential deductions are straightforward generalizations of the standard \mathcal{NDC} semantics [3]. We illustrate here with the axiom for **left-and** and the rule for **assume**, omitting the rest:

$$\frac{}{(\beta \cup \{F \wedge G\}; (\sigma; \rho)) \vdash \mathbf{left-and} \ F \wedge G \rightsquigarrow F}$$

$$\frac{(\beta \cup \{F\}; (\sigma; \rho)) \vdash D \rightsquigarrow G}{(\beta; (\sigma; \rho)) \vdash \mathbf{assume} \ F \ D \rightsquigarrow F \Rightarrow G}$$

The only new sentential form is **observe**, whose semantics are as follows:

$$\frac{}{(\beta; (\sigma; \rho)) \vdash \mathbf{observe} \ F \rightsquigarrow F}$$

provided that $I_{(\sigma; \rho), \emptyset}(F) = \mathbf{true}$

Note that the empty variable assignment is used to evaluate the observed formula, since only sentences can be directly observed—variables do not appear in diagrams.

There are four types of case reasoning in \mathcal{DNDL} :

Sentential-to-sentential: In this type of reasoning we note that a disjunction $F_1 \vee F_2$ holds and that a formula G is entailed in either case, which entitles us to conclude G . This is captured syntactically as a rule application:

$$\mathbf{cases} \ F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G$$

The semantics of such rule applications carry over from \mathcal{NDC} unchanged, since there is no diagram manipulation involved:

$$\frac{}{(\beta \cup \{F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G\}; (\sigma; \rho)) \vdash \mathbf{cases} \ F_1 \vee F_2, F_1 \Rightarrow G, F_2 \Rightarrow G}$$

$$\begin{array}{c}
\frac{(\beta \cup \{F_1, \dots, F_n\}; (\sigma; \rho)) \vdash (\sigma'; \rho') \text{ by thinning with } F_1, \dots, F_n \rightsquigarrow (\sigma'; \rho')}{\text{provided } (\sigma; \rho) \models_{\{F_1, \dots, F_n\}} (\sigma'; \rho')} \quad [\textit{Thinning}] \\
\\
\frac{(\beta \cup \{F_1, \dots, F_n\}; (\sigma; \rho)) \vdash (\sigma'; \rho') \text{ by widening } \rightsquigarrow (\sigma'; \rho')}{\text{provided } (\sigma; \rho) \sqsubseteq (\sigma'; \rho')} \quad [\textit{Widening}] \\
\\
\frac{(\beta \cup \{\text{false}\}; (\sigma; \rho)) \vdash (\sigma'; \rho') \text{ by absurdity } \rightsquigarrow (\sigma'; \rho')}{\quad} \quad [\textit{Absurdity}] \\
\\
\frac{(\beta; (\sigma; \rho)) \vdash \text{claim } (\sigma; \rho) \rightsquigarrow (\sigma; \rho)}{\quad} \quad [\textit{Diagram-Reiteration}] \\
\\
\frac{(\beta \cup \{F_1, \dots, F_k\}; (\sigma_1; \rho_1)) \vdash \Delta_1 \rightsquigarrow (\sigma'; \rho')}{\vdots} \\
\frac{(\beta \cup \{F_1, \dots, F_k\}; (\sigma_n; \rho_n)) \vdash \Delta_n \rightsquigarrow (\sigma'; \rho')}{(\beta \cup \{F_1, \dots, F_k\}; (\sigma; \rho)) \vdash \text{cases from } F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow \Delta_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow \Delta_n \rightsquigarrow (\sigma'; \rho')} \quad [\textit{C1}] \\
\text{provided } (\sigma; \rho) \models_{\{F_1, \dots, F_k\}} \{(\sigma_1; \rho_1), \dots, (\sigma_n; \rho_n)\} \\
\\
\frac{(\beta \cup \{F_1\}; (\sigma; \rho)) \vdash \Delta_1 \rightsquigarrow (\sigma'; \rho') \quad (\beta \cup \{F_2\}; (\sigma; \rho)) \vdash \Delta_2 \rightsquigarrow (\sigma'; \rho')}{(\beta; (\sigma; \rho)) \vdash \text{cases } F_1 \vee F_2 \quad F_1 \rightarrow \Delta_1 \mid F_2 \rightarrow \Delta_2 \rightsquigarrow (\sigma'; \rho')} \quad [\textit{C2}] \\
\\
\frac{(\beta; (\sigma; \rho)) \vdash D \rightsquigarrow F \quad (\beta \cup \{F\}; (\sigma; \rho)) \vdash \Delta \rightsquigarrow (\sigma'; \rho')}{(\beta; (\sigma; \rho)) \vdash D; \Delta \rightsquigarrow (\sigma'; \rho')} \quad [\textit{SD}] \\
\\
\frac{(\beta; (\sigma; \rho)) \vdash \Delta \rightsquigarrow (\sigma'; \rho') \quad (\beta; (\sigma'; \rho')) \vdash D \rightsquigarrow F}{(\beta; (\sigma; \rho)) \vdash \Delta; D \rightsquigarrow F} \quad [\textit{DS}] \\
\\
\frac{(\beta; (\sigma; \rho)) \vdash \Delta_1 \rightsquigarrow (\sigma_1; \rho_1) \quad (\beta; (\sigma_1; \rho_1)) \vdash \Delta_2 \rightsquigarrow (\sigma_2; \rho_2)}{(\beta; (\sigma; \rho)) \vdash \Delta_1; \Delta_2 \rightsquigarrow (\sigma_2; \rho_2)} \quad [\textit{DD}] \\
\\
\frac{(\beta; (\sigma; \rho)) \vdash D_1 \rightsquigarrow F_1 \quad (\beta \cup \{F_1\}; (\sigma; \rho)) \vdash D_2 \rightsquigarrow F_2}{(\beta; (\sigma; \rho)) \vdash D_1; D_2 \rightsquigarrow F_2} \quad [\textit{SS}] \\
\\
\frac{(\beta \cup \{F[x \mapsto z]\}; (\sigma; \rho)) \vdash \Delta \rightsquigarrow (\sigma'; \rho')}{(\beta; (\sigma; \rho)) \vdash \text{pick-witness } w \text{ for } \exists x. F \quad \Delta \rightsquigarrow (\sigma'; \rho')} \quad [\textit{ExInst}] \\
\text{provided } z \text{ is fresh}
\end{array}$$

Figure 1.3: Formal semantics of diagrammatic deductions

Sentential-to-diagrammatic: Here we note that a disjunction $F_1 \vee F_2$ holds and proceed to show that a certain diagram $(\sigma; \rho)$ follows in either case. This is captured by the syntax form

$$\text{cases } F_1 \vee F_2: F_1 \rightarrow \Delta_1 \mid F_2 \rightarrow \Delta_2$$

which is classified as a diagrammatic deduction (“a Δ ”) since the end result is a diagram. The semantics of this form are given by rule $[C_2]$, shown in Figure 1.3.

Diagrammatic-to-sentential: We note that on the basis of the present system state and some formulas F_1, \dots, F_k in the assumption base, one of $n > 0$ other system states $(\sigma_1; \rho_1), \dots, (\sigma_n; \rho_n)$ must obtain, and proceed to show that a formula F can be derived in any one of these n cases. This entitles us to infer F , provided of course that the n diagrammatic cases are indeed exhaustive. This form of reasoning is captured by the form

$$\text{cases from } F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow D_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow D_n$$

This is classified as a sentential deduction, since the end result is a formula F . Its semantics are given by the following rule:

$$\boxed{\begin{array}{c} (\beta \cup \{F_1, \dots, F_k\}; (\sigma_1; \rho_1)) \vdash D_1 \rightsquigarrow F \\ \vdots \\ (\beta \cup \{F_1, \dots, F_k\}; (\sigma_n; \rho_n)) \vdash D_n \rightsquigarrow F \\ \hline (\beta \cup \{F_1, \dots, F_k\}; (\sigma; \rho)) \vdash \text{cases from } F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow D_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow D_n \rightsquigarrow F \\ \text{provided } (\sigma; \rho) \models_{\{F_1, \dots, F_k\}} \{(\sigma_1; \rho_1), \dots, (\sigma_n; \rho_n)\} \end{array}} \quad [C_3]$$

Observe that the caveat that the diagrams $(\sigma_1; \rho_1), \dots, (\sigma_n; \rho_n)$ form an exhaustive set of possibilities on the basis of F_1, \dots, F_k and the current diagram is formally captured by the proviso

$$(\sigma; \rho) \models_{\{F_1, \dots, F_k\}} \{(\sigma_1; \rho_1), \dots, (\sigma_n; \rho_n)\}$$

Diagrammatic-to-diagrammatic: This is similar to the above mode of reasoning, with the exception that instead of deriving a formula F in each of the n cases, we derive a diagram. Therefore, syntactically, following each of the n cases we have diagrammatic deductions $\Delta_1, \dots, \Delta_n$ (rather than sentential deductions D_1, \dots, D_n as we did above), and the entire form is classified as a diagrammatic deduction, since the final conclusion is a diagram. The following syntax form is used for such deductions:

$$\text{cases from } F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow \Delta_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow \Delta_n$$

The corresponding semantics are given by rule $[C_1]$, shown in Figure 1.3.

Likewise, there are four types of deduction compositions:

1. $D_1; D_2$, where a sentential deduction D_1 is composed with another sentential deduction D_2 . This form is classified as a sentential deduction, since the end result is a formula (the conclusion of D_2). Its semantics are given by rule $[SS]$ of Figure 1.3. They are isomorphic to the regular composition semantics of $\mathcal{N}\mathcal{D}\mathcal{L}$, since there is no diagram manipulation involved.

2. $D; \Delta$, where a sentential deduction D is composed with a diagrammatic deduction. This form is classified as a diagrammatic deduction since the end result is a diagram—the conclusion of Δ . Its semantics are prescribed by rule $[SD]$. Observe that the conclusion of D becomes available to Δ (e.g., the conclusion of D could be a disjunction and Δ might be a diagrammatic case analysis of that disjunction).
3. $\Delta; D$, where a diagrammatic deduction Δ is composed with a sentential deduction. This form is classified as a sentential deduction since the end result is a formula (the conclusion of D). Its semantics are given by rule $[DS]$. Conclusion threading here is also intuitive: D will be evaluated in the system state resulting from the evaluation of Δ . E.g., D might be an **observe** deduction that points out something that can be seen in the diagram derived by Δ .
4. $\Delta_1; \Delta_2$, where a diagrammatic deduction Δ_1 is composed with another diagrammatic deduction Δ_2 . This form is of course classified as a diagrammatic deduction, since the end result is a diagram (the conclusion of Δ_2). Its semantics are given by rule $[DD]$. The same principle of conclusion threading applies here: Δ_2 is evaluated in the system state resulting from the evaluation of Δ_1 (the assumption base gets threaded unchanged).

The composition operator “;” associates to the right, so that $\mathbf{D}_1; \mathbf{D}_2; \mathbf{D}_3$ stands for $\mathbf{D}_1; (\mathbf{D}_2; \mathbf{D}_3)$ rather than $(\mathbf{D}_1; \mathbf{D}_2); \mathbf{D}_3$.

The semantics of diagrammatic deductions are shown in Figure 1.3. Note that there are four rules for the composition operator, one for each way of combining a sentential with a diagrammatic deduction. The rule of the fourth case, that of combining two sentential deductions (rule $[SS]$), is isomorphic to the regular composition semantics of \mathcal{NDC} .

Theorem 1.4 (Soundness) *If $(\beta; (\sigma; \rho)) \vdash \mathbf{D} \rightsquigarrow F$ then $\beta \models F$; and if $(\beta; (\sigma; \rho)) \vdash \mathbf{D} \rightsquigarrow (\sigma'; \rho')$ then $(\sigma; \rho) \models_{\beta} (\sigma'; \rho')$.*

Proof: By induction on the structure of \mathbf{D} . ■

Example 1.11 Consider the \mathcal{DNDL} language obtained by fixing the “clock” signature, attribute structure and interpretation of Example 1.9. Now consider a system of two clocks c_1 and c_2 , to which we will give the names \mathbf{c}_1 and \mathbf{c}_2 (recall that c_1 and c_2 are constant symbols of the signature, so this is a constant assignment ρ , which need only be partial). Now let σ be the state depicted by the following picture:



Intuitively, this state signifies that we know the precise time displayed by c_2 (namely, 5:45am). We are also sure of the minute value of c_2 (28), but not of its hour value, which could be either 4, or 5, or 6. Now suppose that we are further given the premise $\mathbf{Ahead}(\mathbf{c}_1, \mathbf{c}_2)$, indicating that the time displayed by c_1 is “ahead” of that displayed by c_2 .

From these two pieces of information, one diagrammatic and the other sentential, we should be able to infer the following diagram, call it σ' :

6:28
c₁

5:45
c₂

That is, we should be able to conclude the exact time of c_1 , since, given that c_1 is ahead of c_2 , the hour displayed by c_1 cannot possibly be 4 or 5—it must, therefore, be 6. We can do this in $\mathcal{DN}\mathcal{DL}$ with the following very simple deduction:

$$(\sigma', \rho) \text{ from Ahead}(c_1, c_2)$$

This deduction, when evaluated in the knowledge base $(\{\text{Ahead}(c_1, c_2)\}; (\sigma, \rho))$, will result in state (diagram) σ' . More formally, we have the following judgment:

$$(\{\text{Ahead}(c_1, c_2)\}; (\sigma, \rho)) \vdash (\sigma', \rho) \text{ from Ahead}(c_1, c_2) \rightsquigarrow (\sigma', \rho)$$

by virtue of

$$(\sigma; \rho) \models_{\{\text{Ahead}(c_1, c_2)\}} (\sigma', \rho) \tag{1.7}$$

Note that the constant assignment does not change in the resulting state.

To establish (1.7) rigorously, we must show that for all named states $(\sigma''; \rho'')$ such that

$$\text{Alt}((\sigma; \rho), (\sigma'; \rho), (\sigma''; \rho''))$$

we have

$$I_{(\sigma''; \rho''), \chi}(\text{Ahead}(c_1, c_2)) = \text{false}$$

for all variable assignments χ , according to Definition 1.8. There are five alternative extensions of σ modulo σ' , call them $\sigma_1, \sigma_2, \sigma_3, \sigma_4$, and σ_5 , depending on how we choose to narrow the *hours* values of c_1 (σ' does not change the *minutes* values of c_1 and the *hours* and *minutes* values of c_2 , so any alternative to σ' must preserve those value sets as well; likewise for the constant assignment ρ). The five possibilities are the following:

$$\begin{aligned} \sigma_1 : \text{hours}(c_1) &= \{4, 5\} \\ \sigma_2 : \text{hours}(c_1) &= \{5, 6\} \\ \sigma_3 : \text{hours}(c_1) &= \{4, 6\} \\ \sigma_4 : \text{hours}(c_1) &= 4 \\ \sigma_5 : \text{hours}(c_1) &= 5 \end{aligned}$$

It is straightforward to verify that

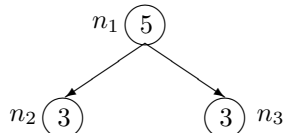
$$I_{(\sigma_1; \rho), \chi}(\text{Ahead}(c_1, c_2)) \neq \text{true}, \dots, I_{(\sigma_5; \rho), \chi}(\text{Ahead}(c_1, c_2)) = \text{false}$$

for all χ . ■

1.5 Representing arbitrary graphs

Graphs (including trees, lists, etc.) are very widely used as diagrammatic depictions of structured data, especially in Computer Science. In this section we present a way of modeling arbitrary graphs in our framework as system states. These ideas will be put to use in the extended example of Section 1.6.

Consider an arbitrary finite graph $G = (N; E)$, where N is a set of nodes and $E \subseteq N \times N$ a set of directed edges. Typically we wish to attach a value to each node $n \in N$, so we assume we have a function $data : N \rightarrow V$ that maps each node to some element of a set of values V . For the purposes of drawing the graph, we also assume that the children of every node are ordered from left to right in some way, i.e., we assume there is a function $children : N \rightarrow N^*$, where N^* is the set of all finite sequences made up of elements of N (arbitrary sequences can be chosen if the ordering is immaterial for displaying the graph). Consider, for instance, the graph:



Here $N = \{n_1, n_2, n_3\}$ and $E = \{(n_1, n_2), (n_1, n_3)\}$. The values attached to the nodes are natural numbers. So we can represent the graph by the functions $data$ and $children$ as mentioned above, where

$$data(n_1) = 5, data(n_2) = 3, data(n_3) = 3$$

and

$$children(n_1) = [n_2, n_3], children(n_2) = [], children(n_3) = []$$

(This is similar to the usual “adjacency list” representation of graphs [13].)

Any graph $G = (N; E)$ where the nodes take values from a set V gives rise to systems of the form $\mathcal{S}_N = (N; \mathcal{A}_N)$, where \mathcal{A}_N is an automorphic attribute structure of the form

$$\mathcal{A}_N = (id : N, children : N^*, data : V; \mathcal{R})$$

Here the attributes $children$ and $data$ are as discussed above, id is the identity function on N , and $D(R) \subseteq \{N, N^*, V\}$ for each relation $R \in \mathcal{R}$ (the precise contents of \mathcal{R} will vary). The graph G itself can be represented as a world of the system \mathcal{S}_N . “Incomplete” graphs where the value and/or children of some nodes are not precisely known can be represented by partial states of such systems.

1.6 Another example: the Mergesort puzzle

In this section we present a more involved $\mathcal{DN}\mathcal{DL}$ by way of a puzzle. In its general form, the puzzle can be described as follows. The output of an algorithm is displayed at the bottom of a diagram depicting a call graph for a particular run of the algorithm; some sentential information might also be given in addition to the diagram. The objective is to infer what input(s) could possibly have resulted in the depicted call graph, or, more precisely, what inputs are consistent with the given information (the given call graph and sentences). Inference is mostly performed diagrammatically, by deriving a sequence of successive call graphs, by performing case analyses involving such graphs, etc. It will be seen that such graphical proofs are considerably more compact and intuitive than sentential analogues. In the next section we illustrate the puzzle informally with Mergesort, while in Section 1.6.2 we formalize it rigorously as a $\mathcal{DN}\mathcal{DL}$.

1.6.1 Guessing the input of Mergesort

Mergesort is a popular $O(n \log n)$ sorting algorithm. The algorithm works according to the divide-and-conquer paradigm [13]: it successively halves the given list until the original input has been broken into one-element pieces, which are trivially sorted; this is the dividing phase. The small lists

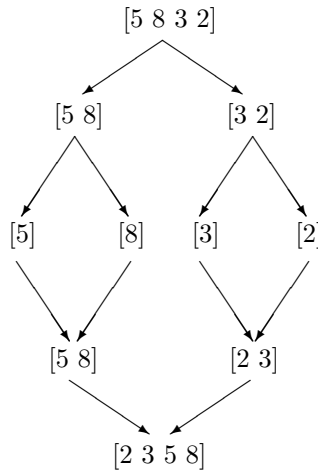


Figure 1.4: The call graph obtained by applying MergeSort to the input list $[5\ 8\ 3\ 2]$.

are then repeatedly combined into larger and larger sorted lists, until we finally obtain the correct sorted permutation of the original input. This is the conquering phase, which turns on the fact that once we have two sorted lists, say $[2\ 8]$ and $[1\ 3\ 5]$, we can efficiently *merge* them to get another sorted list, in this case $[1\ 2\ 3\ 5\ 8]$.

For example, Figure 1.4 depicts the call graph obtained by applying Mergesort to the input list $[5\ 8\ 3\ 2]$. Note that the graph is a DAG (directed acyclic graph). Diverging edges on the top half represent recursive applications of Mergesort to the left and right halves of the input (dividing phase); while converging edges on the lower half represent calls to the merging procedure (conquering phase). For the sake of definitiveness, we make the convention that when the input list is of an odd length $2n + 1$, we take the first n elements as the left half and the remaining $n + 1$ elements as the right half.

The call graph for an application of Mergesort is completely and unambiguously determined once the input list is given. However, things are more interesting in the reverse direction. Clearly, there is no way of retrieving the input list from the output alone, since the inverse of a sorting function is a relation, not a function—any one of $n!$ initial permutations could result in the same sorted n -element list. But if, in addition to specifying the output, we also constrain the call graph of the algorithm by sprinkling some tidbits of information on it or by specifying some sentential information along with it, then we may be able to infer the original input, or at least narrow it down to relatively few possibilities.

As a simple example, suppose you are told that the output of Mergesort is $[1\ 2\ 5\ 8]$. At this point there is not much of interest you can conclude—there are $4! = 24$ possible inputs that could produce this output. But suppose you are further told that the corresponding call graph is as shown in Figure 1.5, where we have attached labels N_i to each node of the graph for easy reference. We write $N_i = ?$ to indicate that we do not know anything about the list that should appear at node N_i ; we write $N_i \supseteq \{x_1, \dots, x_n\}$ to indicate that the numbers x_1, \dots, x_n occur in the said list (though in unknown order, and possibly in tandem with other numbers); and $N_i = [x_1, \dots, x_n]$ to indicate that we know the exact value of the said list to be $[x_1, \dots, x_n]$. From the diagram of Figure 1.5 along with what we know about Mergesort, we can conclude that the original input was either $[2\ 5\ 8\ 1]$ or $[5\ 2\ 8\ 1]$.

The proof consists of two parts: first we derive a sequence of six increasingly detailed diagrams from the given diagram, each extending the previous one, culminating with the diagram in which we

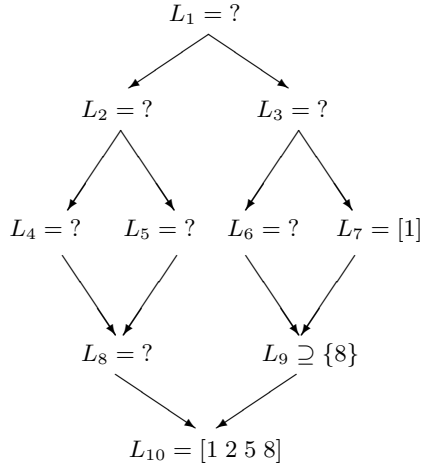


Figure 1.5: A partially unknown MergeSort call graph resulting in the output $[1\ 2\ 5\ 8]$.

know the exact values of all the lists except those of nodes N_1, N_2, N_4 and N_5 ; this part of the proof appears in Figure 1.6. We then perform an exhaustive case analysis by observing that there are only two possibilities at this point: the lists of N_4 and N_5 are (a) $[2]$ and $[5]$, respectively; or else they are (b) $[5]$ and $[2]$, respectively. In the first case we can deduce that the input list was $[2\ 5\ 8\ 1]$, while in the second case we can deduce that it was $[5\ 2\ 8\ 1]$. Therefore, we can infer that the input list was either $[2\ 5\ 8\ 1]$ or $[5\ 2\ 8\ 1]$.

Let us analyze the proof in more detail, beginning with the first part shown in Figure 1.6. That part consists of six steps, labeled (1) through (6). The new information extracted by each step appears in red for enhanced clarity. We discuss each step below:

- Step (1) infers that N_6 must contain the number 8. This follows because we know that 8 occurs in N_9 but not in N_7 ; and that, since N_6 and N_7 converge in N_9 , any number occurs in N_9 iff it occurs either in N_6 or in N_7 (this holds because converging edges indicate list merging).
- Step (2) infers that the list appearing in node N_6 must be precisely $[8]$. We already know from the previous step that 8 occurs in the said list. Now if the list had *any* additional elements, its length would be greater than one, and hence it would be longer than the N_7 list, which we know to have only one element. But this cannot be the case because N_6 and N_7 are the left and right halves of the N_3 list, and every time a list L is split into two halves, the left half is always either of the same length as the right half (if L has even length) or else it is shorter by one (if L has odd length); it cannot possibly be longer. Hence, the N_6 list must be the one-element list $[8]$.
- Step (3) infers that the N_9 list must be $[1\ 8]$. This follows because the N_9 list represents the result of merging N_6 and N_7 , whose precise values are both known at this point.
- Step (4) infers that the N_3 list must be $[8\ 1]$. This follows because we already know the left and right halves of N_3 to be $[8]$ and $[1]$, respectively.
- Step (5) infers that the N_8 list must contain 2 and 5. This holds by virtue of the principle enunciated above in connection with step (1): when L and L' converge in L'' , any number occurs in L'' iff it occurs either in L or in L' . Therefore, since we know that 2 and 5 occur in N_{10} but not in N_9 , they must occur in N_8 .

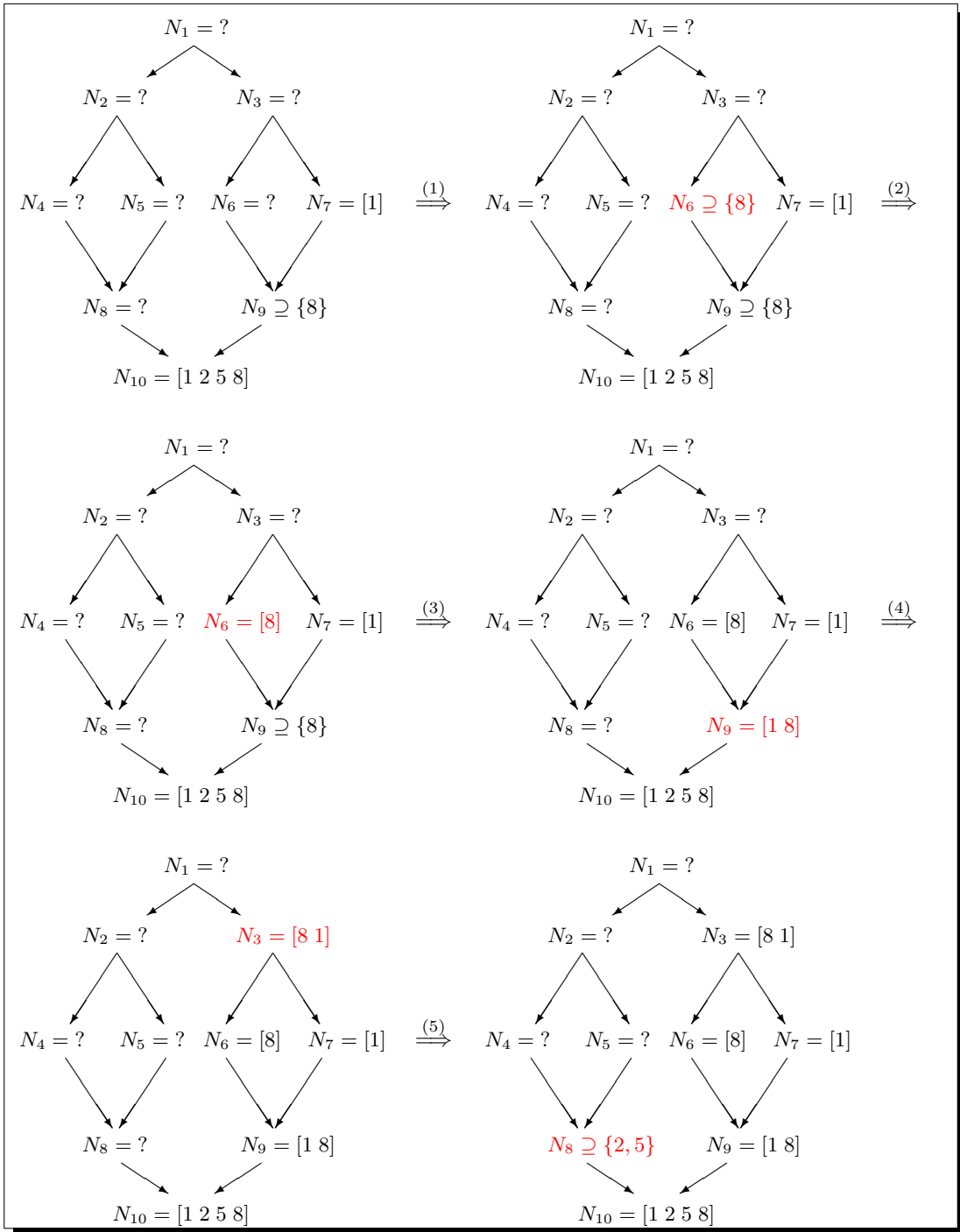


Figure 1.6: First part of a graphical proof solving an instance of the Mergesort puzzle.

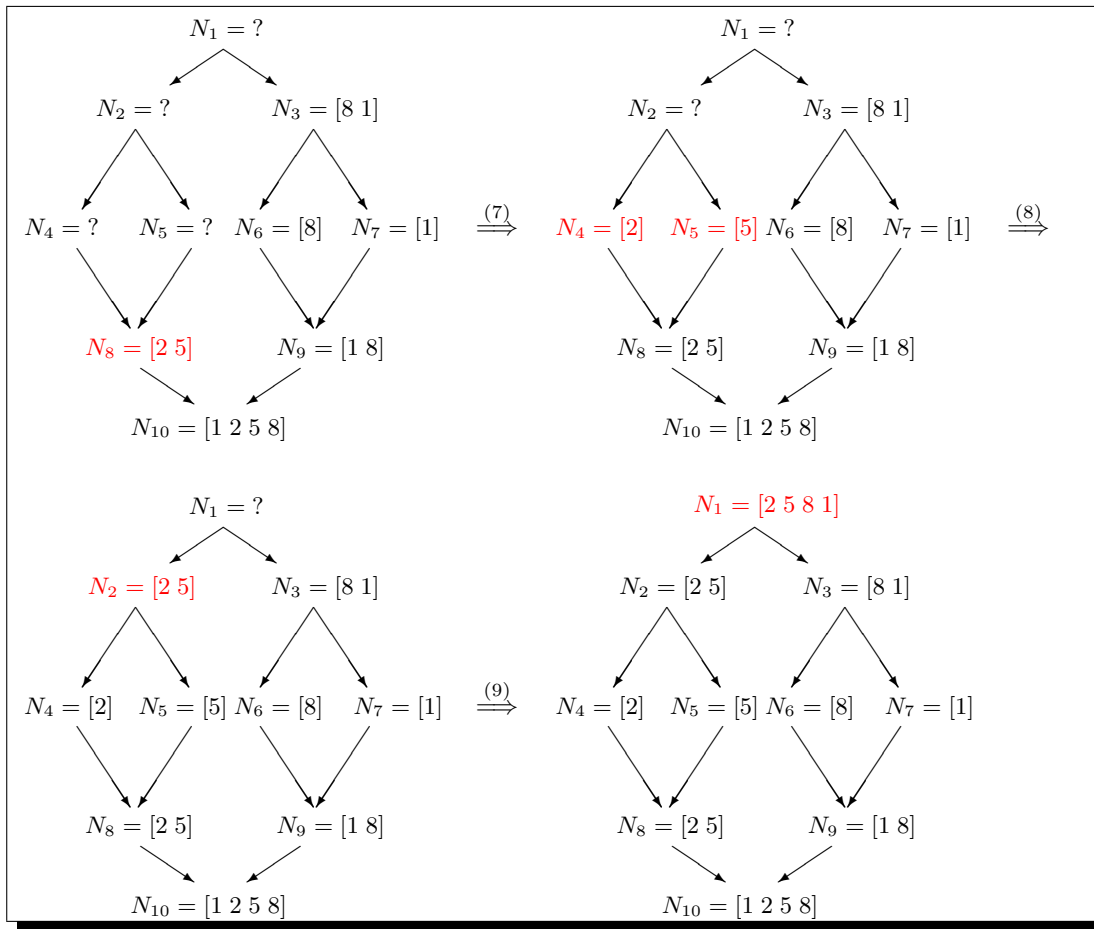


Figure 1.7: Case 1 (out of 2) following the derivation of Figure 1.6.

- Step (6) infers that the N_8 list must be precisely $[2\ 5]$.⁸ We already know that it must have at least these two elements. If it had more than two elements, then N_{10} would have to have at least five elements, given that (a) N_{10} is the result of merging N_8 and N_9 , and that (b) N_9 has two elements. But L_{10} has four elements, therefore 2 and 5 must be the only two elements of N_8 , leaving $[2\ 5]$ and $[5\ 2]$ as the only two possibilities. But the second possibility cannot hold, since N_8 must be sorted (recall that only sorted lists get merged). Hence, the N_8 list must be $[2\ 5]$.

At this point we do not have sufficient information to determine unique values for the N_4 and N_5 lists. However, we can narrow things down to two possibilities: either N_4 and N_5 are $[2]$ and $[5]$, respectively; or else they are $[5]$ and $[2]$. These are the only two possibilities that are consistent with $N_8 = [2\ 5]$, given that N_8 represents the result of merging N_4 and N_5 . The reasoning in each case is

⁸The result of this step does not appear in Figure 1.6 for space reasons, but is shown as the common starting point of the subsequent case analysis depicted by Figure 1.7 and Figure 1.8.

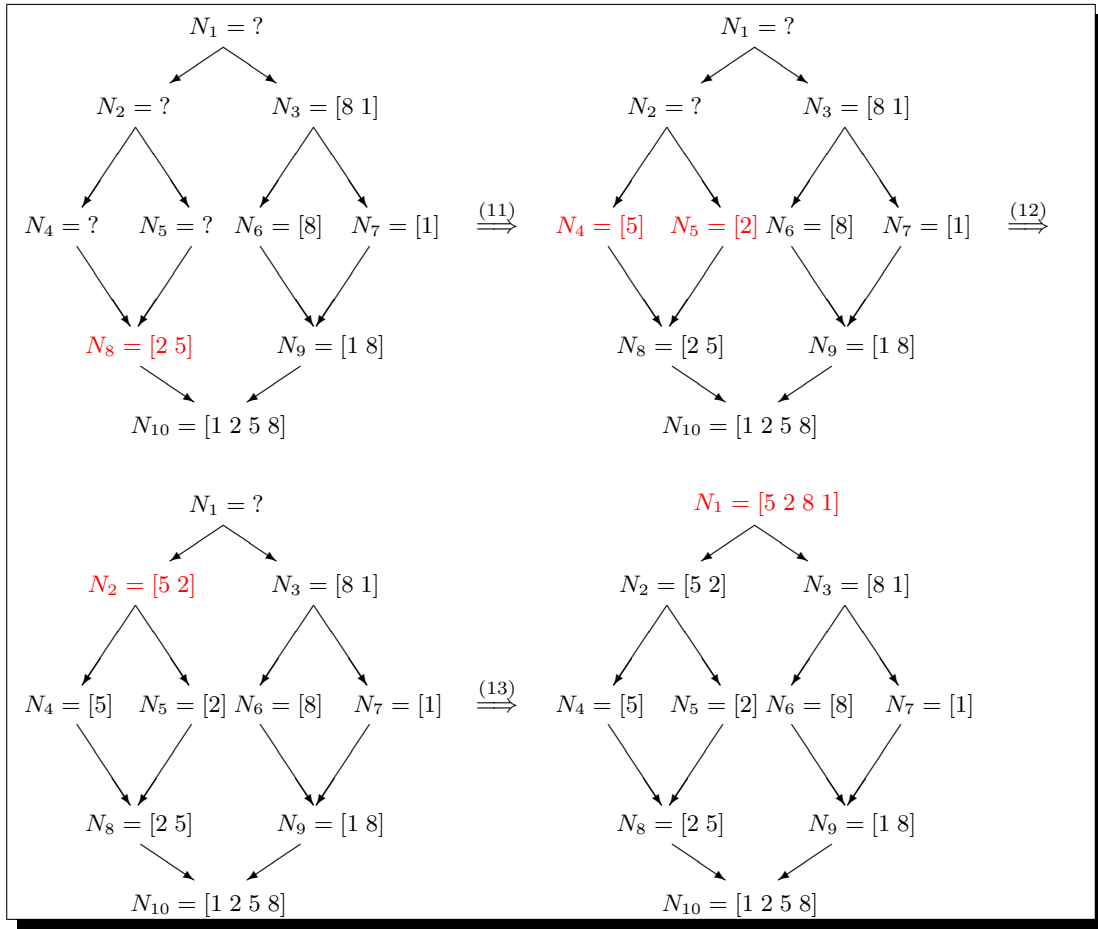


Figure 1.8: Case 2 (out of 2) following the derivation of Figure 1.6.

as follows:

Case 1 : In that case (Figure 1.7), we proceed to infer that the value of N_2 must be $[2\ 5]$, since N_4 and N_5 are the left and right halves of N_2 . And then, since we know both N_2 and N_3 we can determine the value of the input N_1 to be $[2\ 5\ 8\ 1]$.

Case 2 : In that case (Figure 1.8), we deduce that the value of N_2 must be $[5\ 2]$, for the same reason we cited in the preceding case. Similarly, we can then conclude that the input list must be $[5\ 2\ 8\ 1]$.

We are now entitled to infer that the original input list must be either $[2\ 5\ 8\ 1]$ or $[5\ 2\ 8\ 1]$.

1.6.2 Formalizing the puzzle as a $\mathcal{DN}\mathcal{DL}$

There are three steps to obtaining a particular $\mathcal{DN}\mathcal{DL}$:

1. Specify an attribute structure \mathcal{A} .
2. Specify a vocabulary Σ .
3. Specify an interpretation of the relation symbols of Σ into \mathcal{A} as discussed in Section 1.3.

In the following three sections we carry out these three steps in detail for the Mergesort puzzle.

Specifying the attribute structure

Let *Node* be the universe of nodes and let Z^* be the set of all finite sequences (lists) of integers. An appropriate attribute structure for the Mergesort puzzle is the following:

$$\mathcal{A}_M = (id : Node, children : Node^*, data : Z^*; \{R_1, R_2, R_3, R_4, R_5, R_6, R_7\} \cup \{R_L \mid L \in Z^*\})$$

where the relations R_1, \dots, R_6, R_L are as follows:

- $R_1 \subseteq Node^* \times Node \times Node$, with

$$R_1([n_1 \dots n_k], n, n') \Leftrightarrow \{n, n'\} \subseteq \{n_1, \dots, n_k\}$$

- $R_2 \subseteq Node \times Node^* \times Node^*$, with

$$R_2(n, [n_1 \dots n_k], [n'_1 \dots n'_m]) \Leftrightarrow n \in \{n_1, \dots, n_k\} \cap \{n'_1, \dots, n'_m\}$$

- $R_3 \subseteq Z^* \times Z^* \times Z^*$, with

$$R_3([x_1 \dots x_k], [y_1 \dots y_n], [z_1 \dots z_m]) \Leftrightarrow [x_1 \dots x_k] = [y_1 \dots y_n z_1 \dots z_m]$$

i.e., iff $[x_1 \dots x_k]$ is the concatenation of $[y_1 \dots y_n]$ and $[z_1 \dots z_m]$.

- $R_4 \subseteq Z^* \times Z^*$, with

$$R_4([x_1 \dots x_k], [y_1 \dots y_n]) \Leftrightarrow n \in \{k, k+1\}$$

- $R_5 \subseteq Z^*$, with

$$R_5([x_1 \dots x_k]) \Leftrightarrow x_i \leq x_{i+1} \text{ for } i = 1, \dots, k-1$$

i.e., iff $[x_1 \dots x_k]$ is sorted

- $R_6 \subseteq Z^* \times Z^* \times Z^*$, with

$$R_6([x_1 \dots x_k], [y_1 \dots y_n], [z_1 \dots z_m]) \Leftrightarrow \{x_1, \dots, x_k\} = \{y_1, \dots, y_n\} \cup \{z_1, \dots, z_m\}$$

- $R_7 \subseteq Z^* \times Z^* \times Z^*$, with

$$R_7([x_1 \dots x_k], [y_1 \dots y_n], [z_1 \dots z_m]) \Leftrightarrow k = n + m$$

- $R_L \subseteq Z^*$, with

$$R_{[x_1 \dots x_k]}([y_1 \dots y_n]) \Leftrightarrow [x_1 \dots x_k] = [y_1 \dots y_n]$$

Note that we have infinitely many unary relations R_L , parameterized by L . Each such relation takes an arbitrary list of integers L' and tests for the equality $L' = L$.

To make things concrete, Figure 1.10 presents an implementation of this attribute structure in SML.

Specifying the vocabulary

We have seven relations symbols: **peak**, **valley**, **append**, **union**, and **sum** are ternary; **halves** is binary; and **sorted** is unary. In addition, for each list of integers L we have a unary relation symbol val_L . We use N_1, N_2, \dots as constant symbols and v_1, v_2, \dots as variables.

Specifying the interpretation

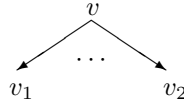
The interpretation of the relation symbols is shown in Figure 1.9.

Symbol	Arity	Realization	Profile
peak	3	R_1	$[(children, 1), (id, 2), (id, 3)]$
valley	3	R_2	$[(id, 1), (children, 2), (children, 3)]$
append	3	R_3	$[(data, 1), (data, 2), (data, 3)]$
halves	2	R_4	$[(data, 1), (data, 2)]$
sorted	1	R_5	$[(data, 1)]$
union	3	R_6	$[(data, 1), (data, 2), (data, 3)]$
sum	3	R_7	$[(data, 1), (data, 2), (data, 3)]$
val_L	1	R_L	$[(data, 1)]$

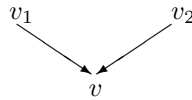
Figure 1.9: The interpretation of the puzzle vocabulary.

More intuitive explanations follow:

- $\text{peak}(v, v_1, v_2)$ holds iff nodes v_1 and v_2 are both children of v :



- $\text{valley}(v, v_1, v_2)$ holds iff nodes v_1 and v_2 are both parents of v :



- $\text{append}(v_1, v_2, v_3)$ holds iff the list attached to node v_1 (i.e., the *data* field of v_1) is identical to the concatenation of the lists attached to nodes v_2 and v_3 , respectively.
- $\text{halves}(v_1, v_2)$ holds iff the lengths of the lists attached to nodes v_1 and v_2 are approximately equal; or, more precisely, iff the length of the v_2 list is either equal or one more than the length of the v_1 list.
- $\text{sorted}(v)$ holds iff the list attached to node v is sorted.
- $\text{union}(v_1, v_2, v_3)$ holds iff the list attached to node v_1 contains all and only those elements that occur either in v_2 or in v_3 (or in both).
- $\text{sum}(v_1, v_2, v_3)$ holds iff the length of the v_1 list is equal to the sum of the lengths of the v_2 and v_3 lists.

```

datatype Nat = zero | succ of Nat;

datatype Node = node of Nat;

fun member(x,L) = List.exists (fn y => x = y) L;

fun subset(L1,L2) = List.all (fn x => member(x,L2)) L1;

fun R1(L,n1,n2) = member(n1,L) andalso member(n2,L);

fun R2(n,L1,L2) = member(n,L1) andalso member(n,L2);

fun R3(L1,L2,L3) = L1 = L2@L3;

fun R4(L1,L2) = let val len1 = length L1
                  val len2 = length L2
                in
                  len2 = len1 orelse len2 = len1 + 1
                end;

fun R5([]) = true
  | R5(x::L) = R5(L) andalso List.all (fn y => x <= y) L;

fun R6(L1,L2,L3) = let val L = L2@L3
                     in
                       subset(L1,L) andalso subset(L,L1)
                     end;

fun R7(L1,L2,L3) = length(L1) = length(L2) + length(L3);

```

Figure 1.10: SML code implementing the attribute structure of the MergeSort puzzle.

- $\text{val}_L(v)$ holds iff the list attached to node v is identical to L . We write $\text{val}(v, L)$ as an abbreviation for $\text{val}_L(v)$.

Axiomatization

The following Horn clauses are all the axioms we need for solving Mergesort puzzles. Their meaning should be clear in light of the foregoing interpretation.

$$\begin{array}{ll}
\forall v, v_1, v_2 . \text{peak}(v, v_1, v_2) \Rightarrow \text{halves}(v_1, v_2) & [\text{halves-axiom}] \\
\forall v, v_1, v_2 . \text{valley}(v, v_1, v_2) \Rightarrow \text{sorted}(v_1) \wedge \text{sorted}(v_2) \wedge \text{sorted}(v) & [\text{sorted-axiom}] \\
\forall v, v_1, v_2 . \text{valley}(v, v_1, v_2) \vee \text{peak}(v, v_1, v_2) \Rightarrow \text{union}(v, v_1, v_2) & [\text{union-axiom}] \\
\forall v, v_1, v_2 . \text{peak}(v, v_1, v_2) \Rightarrow \text{append}(v, v_1, v_2) & [\text{append-axiom}] \\
\forall v, v_1, v_2 . \text{valley}(v, v_1, v_2) \Rightarrow \text{sum}(v, v_1, v_2) & [\text{sum-axiom}]
\end{array}$$

1.6.3 The formal proof

Let $node_1, \dots, node_{10}$ be ten nodes from the universe of all nodes, $Node$. In combination with the attribute structure \mathcal{A}_M defined earlier, these ten nodes constitute a system. The diagrams shown in Figure 1.6, Figure 1.7 and Figure 1.8 depict specific named states of this system. Consider, for

instance, the starting diagram, at the upper left corner of Figure 1.6. This represents a named state $(\sigma; \rho)$, where the partial constant assignment ρ is

$$N_1 \mapsto \text{node}_1, N_2 \mapsto \text{node}_2, \dots, N_{10} \mapsto \text{node}_{10} \quad (1.8)$$

(with $\rho(N_i)$ undefined for $i > 10$); while the two ascriptions *children* and *data* are as follows (the *id* ascription is defined in the obvious way):

$$\begin{aligned} \text{children}(\text{node}_1) &= [\text{node}_2, \text{node}_3] \\ &\vdots \\ \text{children}(\text{node}_5) &= [\text{node}_8] \\ &\vdots \\ \text{children}(\text{node}_{10}) &= [] \end{aligned}$$

and

$$\text{data}(\text{node}_1) = \{[], [1], [2], [5], [8], [1\ 2], [1\ 5], \dots, [8\ 5\ 1], \dots, [1\ 2\ 5\ 8]\} \quad (1.9)$$

$$\begin{aligned} &\vdots \\ \text{data}(\text{node}_9) &= \{[8], [1\ 8], [8\ 1], [2\ 8], \dots, [5\ 8\ 1], [2\ 5\ 1\ 8], \dots\} \end{aligned} \quad (1.10)$$

$$\begin{aligned} &\vdots \\ \text{data}(\text{node}_{10}) &= \{[1\ 2\ 5\ 8]\} \end{aligned} \quad (1.11)$$

Observe (1.9). At this point we do not know anything about what list appears at *node*₁ (a complete lack of knowledge signified by the inscription $N_1 = ?$), so the data field of *node*₁ is entirely unconstrained: it contains all possible lists of length four obtained by permutations of four objects taken four at a time ($P(4, 4) = 4! = 24$ total); plus all possible lists of length three obtained by permutations of four objects taken three at a time ($P(4, 3) = 24$ total); plus all possible lists of length two obtained by permutations of four objects taken two at a time ($P(4, 2) = 12$), plus all possible lists of length one (4), plus the empty list, for a sum total of $24 + 24 + 12 + 4 + 1 = 65$ different lists. The *data* ascription maps every “questionmark node” (e.g., the nodes labeled by N_6 or N_8) to the same set of 65 lists. Hereafter we will denote this set of 65 lists by \mathcal{L} . By contrast, the *data* ascription for *node*₉ (the node labeled by N_9) is subject to the constraint that all list values must contain 8, so this narrows down the possibilities to a total of $24 + 18 + 6 + 1 = 49$. Further down, the value of *data* for *node*₁₀ is completely determined—the singleton $\{[1\ 2\ 5\ 8]\}$.⁹ The named system state corresponding to any of the diagrams shown in connection with the Mergesort puzzle is likewise defined. The *children* ascription and the constant assignment remain the same in every case; while the *data* value is specified in accordance with the preceding conventions.

Extracting the appropriate system state from a given diagram can be viewed as the task of computing a parsing function ϕ that takes a concrete two-dimensional representation and produces an

⁹These are unnecessarily coarse approximations. We could leverage our knowledge of the domain to further cut down the possibilities drastically. For instance, we know that at the top node only lists of length four could appear—or, in general, only lists of the exact same length as the unique list that appears at the bottom node representing the output. Further, we know that if any node has only lists of n items as possible values, then the left and right children can respectively only have lists of length $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ as possible values, and so on. In this manner cardinality constraints would propagate down the graph and significantly curtail the values of the *data* ascription. This would be important for an efficient implementation of the Mergesort puzzle as a $\mathcal{DN}\mathcal{DL}$, but it is not necessary for the purposes of this paper.

```

 $\tau_2$  by thinning with union-axiom;
 $\tau_3$  by thinning with halves-axiom;
 $\tau_4$  by thinning with union-axiom, sorted-axiom;
 $\tau_5$  by thinning with append-axiom;
 $\tau_6$  by thinning with union-axiom;
 $\tau_7$  by thinning with sum-axiom, sorted-axiom;
cases from union-axiom, halves-axiom:
   $\tau_8 \rightarrow \tau_9$  by thinning with append-axiom;
     $\tau_{10}$  by thinning with append-axiom;
    observe  $\text{val}(N_1, [2\ 5\ 8\ 1]) \vee \text{val}(N_1, [5\ 2\ 8\ 1])$ 
 $\tau_{12} \rightarrow \tau_{13}$  by thinning with append-axiom;
   $\tau_{14}$  by thinning with append-axiom;
  observe  $\text{val}(N_1, [2\ 5\ 8\ 1]) \vee \text{val}(N_1, [5\ 2\ 8\ 1])$ 

```

Figure 1.11: Formal $\mathcal{DN}\mathcal{DL}$ proof solving the Mergesort puzzle of Section 1.6.1.

abstract syntax tree for it. Conversely, reconstructing a diagram from the underlying system state can be seen as computing an “unparsing” function ψ that proceeds in the reverse direction, rendering system states graphically. As with customary parsing and unparsing, we have

$$\psi(\phi(d)) = d \text{ and } \phi(\psi(\sigma)) = \sigma \quad (1.12)$$

for all Mergesort puzzle diagrams d and system states σ , where the first identity is understood to obtain up to topological equivalence.¹⁰ From a practical standpoint, most of the effort required to build a $\mathcal{DN}\mathcal{DL}$ would be allotted to the implementation of these two functions. In the case of the Mergesort puzzle, both ϕ and ψ can be computed efficiently—in low polynomial time—using standard graph-theoretic algorithms.

Finally, Figure 1.11 shows the formal $\mathcal{DN}\mathcal{DL}$ proof that solves the Mergesort puzzle discussed in Section 1.6.1. We conclude with a detailed analysis of this proof.

First, we need a very simple lemma: the formula

$$\forall v, v_1, v_2 . \text{valley}(v, v_1, v_2) \Rightarrow \text{union}(v, v_1, v_2) \wedge \text{sum}(v, v_1, v_2) \quad [\textit{lemma}]$$

This can be derived from our five axioms in a few lines of $\mathcal{DN}\mathcal{DL}$, by some elementary sentential reasoning; we leave the derivation to the reader.

Next, let $\sigma_1, \dots, \sigma_6$ be the system states corresponding to the six diagrams that appear in Figure 1.8 starting from the top left corner and moving clockwise, so that σ_i represents the graph to the left of the arrow indicating the i^{th} step. Likewise, let $\sigma_7, \dots, \sigma_{10}$ and $\sigma_{11}, \dots, \sigma_{14}$ be the states corresponding to the diagrams of Figure 1.7 and Figure 1.8, respectively. For any $i = 1, \dots, 14$, we write τ_i to denote the named state $(\sigma_i; \rho)$, where ρ is the constant assignment (1.8).

¹⁰Diagrammatic identity in general can be a vague notion (e.g., when exactly can we say that two drawings depict the same mountain range?) and this is part of the reason why logicians and mathematicians have had a skeptical attitude towards diagrams (Quine’s dictum “No entity without identity” [35] comes to mind). Nevertheless, there are many cases where we can formulate rigorous necessary and sufficient conditions for two diagrams to be considered identical, using topological or other extensional notions. Alternatively, it would not be unreasonable to define diagram identity constructively, as identity on the corresponding system states produced by a parser of the kind discussed above, although that would render equations such as (1.12) tautological.

The proof in Figure 1.11 is a sentential proof D , as it is of the form

$$D = \Delta_1; \dots; \Delta_6; D'$$

i.e., a composition of six diagrammatic steps $\Delta_1, \dots, \Delta_6$ followed by a sentential deduction D' of the form

$$\text{cases from } F_1, \dots, F_k: (\sigma_1; \rho_1) \rightarrow D_1 \mid \dots \mid (\sigma_n; \rho_n) \rightarrow D_n$$

a diagrammatic-to-sentential case analysis (recall that composition is right-associative). The starting point for the proof is the knowledge base

$$\gamma_1 = (\beta_1; \tau_1) \tag{1.13}$$

where β_1 contains the five universally quantified clauses of our axiomatization and the aforementioned lemma. This is the knowledge base in which the entire proof D will be evaluated.

Let us why the first step Δ_1 , the diagrammatic inference

$$\tau_2 \text{ by thinning with } \textit{union-axiom}$$

succeeds. According to the semantics of thinning (Figure 1.3), this step will be valid provided that

$$\tau_1 \models_{\textit{union-axiom}} \tau_2$$

i.e., provided that τ_1 entails τ_2 with respect to *union-axiom*. This means that every alternative way of extending τ_1 modulo τ_2 must falsify *union-axiom* (for an arbitrary variable assignment). More precisely, it must be the case that for every named state $\tau = (\sigma; \rho)$ such that $\textit{Alt}(\tau_1, \tau_2, \tau)$ we have

$$I_{(\sigma; \rho), \chi}(\textit{union-axiom}) = \textbf{false} \tag{1.14}$$

for all χ . Pick any such τ . Since τ_1 and τ_2 share the same constant assignment ρ , the only way τ can be an alternative extension of τ_1 modulo τ_2 is if we have $\textit{Alt}(\sigma_1, \sigma_2, \sigma)$ (Definition 1.7). The only state σ that qualifies as such an alternative is the one that is identical to σ_1 except that the *data* ascription maps \textit{node}_6 to the set of all lists in \mathcal{L} that do *not* contain 8. It is easy to see that (1.14) holds in that state. Indeed, consider an arbitrary χ . By (1.5), *union-axiom* will be false in $(\sigma; \rho)$ and χ if there are some nodes \textit{node}_{i_1} , \textit{node}_{i_2} , and \textit{node}_{i_3} such that

$$I_{(\sigma; \rho), \chi[v \mapsto \textit{node}_{i_1}, v_2 \mapsto \textit{node}_{i_2}, v_3 \mapsto \textit{node}_{i_3}]}(\textbf{valley}(v, v_1, v_2) \vee \textbf{peak}(v, v_1, v_2) \Rightarrow \textbf{union}(v, v_1, v_2)) = \textbf{false}$$

Let these three nodes be \textit{node}_9 , \textit{node}_6 , and \textit{node}_7 , respectively (i.e., the nodes labeled by N_9 , N_6 and N_7). For these nodes we clearly have:

$$I_{(\sigma; \rho), \chi[v \mapsto \textit{node}_9, v_2 \mapsto \textit{node}_6, v_3 \mapsto \textit{node}_7]}(\textbf{valley}(v, v_1, v_2) \vee \textbf{peak}(v, v_1, v_2)) = \textbf{true}$$

(since the nodes form a valley) and yet

$$I_{(\sigma; \rho), \chi[v \mapsto \textit{node}_9, v_2 \mapsto \textit{node}_6, v_3 \mapsto \textit{node}_7]}(\textbf{union}(v, v_1, v_2)) = \textbf{false} \tag{1.15}$$

The reason why (1.15) holds is that for *every* list L in the *data* field of \textit{node}_9 in σ and for every list L' in the *data* field of \textit{node}_6 in σ and every list L'' in the *data* field of \textit{node}_7 in σ , we have

$$\neg R_6(L, L', L'')$$

the reason being that every such L contains 8 but no such L'' contains 8 (because $data(node_7)$ in σ contains only one list value, [1]) and no such L' contains 8 (by virtue of σ being an alternative extension of σ_1 modulo σ_2).

It is important to note that in practice these three nodes would be discovered automatically by exhaustive search. Specifically, the system would evaluate the formula *union-axiom* in the named state $(\sigma; \rho)$ and the empty variable assignment χ ¹¹ to determine if it comes out **false**. Now a universally quantified formula such as *union-axiom* is evaluated in a given χ by binding the universally quantified variable to successive system objects and recursively evaluating the body in the updated χ . If the body comes out **false** for some system object, the whole formula is deemed **false**. If the body itself is another universally quantified formula then we have more choice points and possible backtracking. In the worst case for the puzzle example, the evaluation of *union-axiom* will need to examine $10^3 = 1000$ difference possible assignments of variables to objects, since the system comprises 10 nodes and the formula has three universally quantified variables. The body of *union-axiom* would be evaluated for each of the 1000 node triples. For most of these triples, *union-axiom* would come out **unknown** because there is not enough information to enable a definitive judgment. Consider, for instance, the evaluation of the body of *union-axiom* in the triple

$$\chi[v \mapsto node_1, v_2 \mapsto node_2, v_3 \mapsto node_3]$$

While it is true that $node_1, node_2$, and $node_3$ form a peak, we have

$$I_{(\sigma; \rho), \chi[v \mapsto node_1, v_2 \mapsto node_2, v_3 \mapsto node_3]}(\mathbf{union}(v, v_1, v_2)) = \mathbf{unknown}$$

because, in σ , the realization of **union**, R_6 , holds for some list values in the corresponding *data* fields of $node_1, node_2$ and $node_3$ and does not hold for others (see (1.3)).

In this particular example we have 10 system objects and the most populous attribute value has 65 elements, so a formula such as $\mathbf{union}(v, v_1, v_2)$ could, in theory, take up to $65^3 = 274,625$ evaluations to settle. Combined with the 1,000 triple possibilities dictated by three universal quantifiers, we could look at the non-trivial number of 274,625,000 evaluations. However, in practice atomic formulas such as $\mathbf{union}(v, v_1, v_2)$ would be settled quite speedily because for most node assignments we would get some **true** and some **false** values, quickly leading to an **unknown** result. So the worst case of 1000 different evaluations of atomic formulas is not formidable. Nevertheless, we observe that the user can always improve the efficiency of the proof checking by providing more information in the proof—information that guides the search in the right direction. For example, we could replace the first step

τ_2 **by thinning with** *union-axiom*

by the following sequence of steps:

specialize *union-axiom* **with** N_9, N_6, N_7 ;
observe valley (N_9, N_6, N_7) ;
right-either $\mathbf{peak}(N_9, N_6, N_7) \vee \mathbf{valley}(N_9, N_6, N_7)$;
modus-ponens $\mathbf{peak}(N_9, N_6, N_7) \vee \mathbf{valley}(N_9, N_6, N_7) \Rightarrow \mathbf{union}(N_9, N_6, N_7)$,
 $\mathbf{peak}(N_9, N_6, N_7) \vee \mathbf{valley}(N_9, N_6, N_7)$;
 τ_2 **by thinning with** $\mathbf{union}(N_9, N_6, N_7)$

¹¹It is legitimate to use the empty variable assignment in evaluating a sentence, because, by Lemma 1.1, the truth value of a sentence in the empty variable assignment will be identical to the truth value of that sentence in *every* variable assignment.

Here we focus directly on the three nodes of interest by citing $\mathbf{union}(N_9, N_6, N_7)$ as the justification of the thinning step, instead of citing the universally quantified *union-axiom*. By eliminating the three universal quantifiers, we avert the need to evaluate the body of *union-axiom* over all possible triples of nodes. The tradeoff here is a typical manifestation of the classical tension between brevity and efficiency: a very brief proof takes large steps whose verification can be difficult because it requires search; whereas a detailed proof takes small steps that are easy to check because they involve little or no search. We can always buy efficiency at the expense of conciseness.

Returning to the proof, let us examine the second step:

τ_3 **by thinning with** *halves-axiom*

As with the previous application of thinning, this step is valid only if

$$\tau_2 \models_{\text{halves-axiom}} \tau_3$$

meaning that any named state $\tau = (\sigma; \rho)$ that is an alternative extension of τ_2 modulo τ_3 must falsify *halves-axiom*. As before, because the constant assignment does not change, the only way we can have $\text{Alt}(\tau_2, \tau_3, \tau)$ is if we have $\text{Alt}(\sigma_2, \sigma_3, \sigma)$. And given that in σ_2 the *data* field of *node*₆ contains all and only those lists that contain 8, σ is an alternative extension of σ_2 modulo σ_3 iff it is a list in \mathcal{L} that contains 8 and has length greater than one, e.g., [2 5 8]. But in that state *halves-axiom* is falsified (with *node*₃, *node*₆, and *node*₇ providing the counterexample peak), hence the thinning step is sanctioned. Similar rationales justify the next four thinning steps. We encourage the reader to work through them rigorously.

We come finally to the case analysis, which hinges on the claim that from the state σ_7 and on the basis of the lemma, there are only two possible states, σ_8 and σ_{12} . Symbolically,

$$(\sigma_7; \rho) \models_{\text{lemma}} \{(\sigma_8; \rho), (\sigma_{12}; \rho)\} \tag{1.16}$$

Consulting Definition 1.8, we see that (1.16) holds iff for every $(\sigma'; \rho')$ such that

$$\text{Alt}((\sigma_7; \rho), \{(\sigma_8; \rho), (\sigma_{12}; \rho)\}, (\sigma'; \rho')) \tag{1.17}$$

we have $I_{(\sigma'; \rho'), \chi}(\text{lemma}) = \mathbf{false}$ for all χ . Again, because the constant assignment did not change, (1.17) holds iff

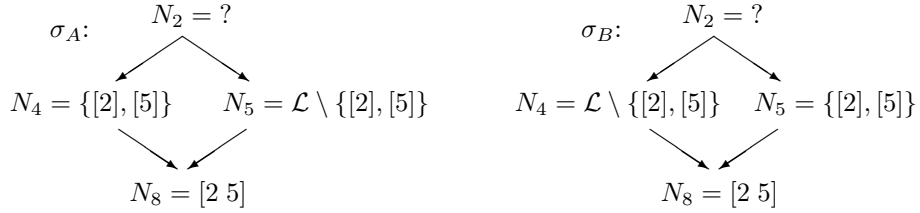
$$\text{Alt}(\sigma_7, \{\sigma_8, \sigma_{12}\}, \sigma') \tag{1.18}$$

(by Definition 1.7). Further, by Definition 1.6, (1.18) holds iff

$$\text{Alt}(\sigma_7, \cup(\sigma_8, \sigma_{12}), \sigma')$$

i.e., iff σ' is an alternative extension of σ_7 modulo the union of σ_8 and σ_{12} . The union in question is the state that assigns $\{[2], [5]\}$ to the *data* fields of both *node*₄ and *node*₅, while all other ascription values stay the same as they are in σ_7 .

Now there are two alternative extensions of σ_7 modulo this union: one, call it σ_A , in which we keep the $\{[2], [5]\}$ value of *node*₄ steady but complement it for *node*₅; and the other, call it σ_B , is one in which we complement the *data* value of *node*₄ in σ_7 and retain the *data* value of *node*₆. The relevant parts of both states can be depicted graphically as follows:



A routine calculation will confirm that both possibilities falsify the cited lemma.

1.7 Conclusions

We have introduced $\mathcal{DN}\mathcal{D}\mathcal{L}$, a family of denotational proof languages (DPLs) that combine sentential and diagrammatic reasoning in Fitch-style natural deduction. Our framework is based on the notions of attribute-structure systems, and on the idea of using Kleene’s three-valued logic to interpret first-order signatures into attribute structures. To obtain a particular instance of $\mathcal{DN}\mathcal{D}\mathcal{L}$, we need only specify an attribute structure, a signature, and an interpretation of the signature into the structure. The result will be a language with a formal abstract syntax and semantics.

We have not discussed how diagrams would be represented within the proof text. This is an interface design issue, not an issue of abstract syntax or semantics. One possibility would be to give names to diagrams and then have those names appear in the proof text, but with hyperlinks. If a user clicks on such a link, a picture depicting the corresponding diagram would pop up, and the user could view or edit the diagram as necessary, save it as another diagram, etc. Of course, how diagrams are drawn depends on the domain at hand; it is completely separate from other aspects of the language. We are designing the implementation of $\mathcal{DN}\mathcal{D}\mathcal{L}$ as an SML functor [32] that will take an attribute structure \mathcal{A} ; the interpretation of a signature Σ into \mathcal{A} ; and a drawing module that can draw an arbitrary \mathcal{A} -system; and will output a parser and an interpreter, i.e., a proof checker for the instantiated language.

Introducing names brings up another intriguing possibility. As it stands, an implementation of $\mathcal{DN}\mathcal{D}\mathcal{L}$ would be a type- α DPL, i.e., a proof checker: it would accept a proof (combining sentential and diagrammatic steps) and would either pronounce it sound or else point out a reasoning error. If we introduce unrestricted naming and computation, we can make these into type- ω DPLs [4, 5], capable not only of proof checking but of arbitrary proof search as well. It would be very interesting to see what types of methods can be written in such a setting for the purpose of automatic diagrammatic inference, and exactly what type of guarantee formal soundness would provide in practice.

Another important issue is efficiency. It is easy to see that depending on the system we are working with, we may need exponential time in the size of the attributes to check whether an application of a rule such as thinning is valid. This is due solely to the size of the attributes and is orthogonal to how “large” are the steps taken by the user. Even if the user takes a very small step, say to exclude one possible value from a set thereof, we may still need to explore exponentially many subsets. Two possibilities for ameliorating this issue are: (a) representing sets of attribute values by binary decision diagrams (BDDs) [10], and (b) symbolic evaluation. For (a), it is hoped that a compact representation of the relevant subsets might speed up rule checking. (There are standard techniques for representing an arbitrary subset S' of a finite set S by a BDD, basically by encoding the characteristic function of S' as a Boolean function [23].) With symbolic evaluation, we may be able to prune very large parts of the search tree if we incorporate a modest degree of domain knowledge into the search process. For instance, if we have determined that a time (h_1, m_1) of a clock c_1 is not ahead of some

clock c_2 , there is no point in trying other possible times (h'_1, m'_1) of c_1 if $h'_1 < h_1$ or if $h'_1 = h_1$ and $m'_1 < m_1$. Sophisticated techniques for performing symbolic predicate evaluation (similar to the symbolic evaluation methods in model checking [12]) could have a dramatic payoff; much work remains to be done here.

Bibliography

- [1] Jaume Agusti, Jordi Puigsegur, and David Stuart Robertson. A visual syntax for logic and logic programming. *Journal of Visual Languages and Computing*, 9(4):399–427, 1998.
- [2] K. Arkoudas. Denotational Proof Languages. PhD dissertation, MIT, 2000.
- [3] K. Arkoudas. Type- α DPLs. MIT AI Memo 2001-25.
- [4] K. Arkoudas. Type- ω DPLs. MIT AI Memo 2001-27.
- [5] T. Arvizo. A virtual machine for a type- ω denotational proof language. Masters thesis, MIT, June 2002.
- [6] J. Barwise and J. Etchemendy. Heterogeneous logic. In J. Glasgow, N. Narayanan, and N. H. Chandrasekaran, editors, *Diagrammatic Reasoning*, pages 211–234. MIT Press, Cambridge, USA, 1995.
- [7] J. Barwise and J. Etchemendy. *Hyperproof: for Macintosh*. CSLI Publications, 1995.
- [8] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.
- [9] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: The see-through interface. *Computer Graphics*, 27(Annual Conference Series):73–80, 1993.
- [10] Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
- [11] S.-K. Chang, editor. *Principles of Visual Programming Systems*. Prentice Hall, New York, 1990.
- [12] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 1999.
- [13] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [14] H. G. Eggleston. *Convexity*. Cambridge University Press, 1969.
- [15] G. Englebretsen. Linear diagrams for syllogisms (with relationals). *Notre Dame Journal of Formal Logic*, 33(1):37–69, 1992.

- [16] David W. Etherington, Alexander Borgida, Ronald J. Brachman, and Henry A. Kautz. Vivid knowledge and tractable reasoning: preliminary report. In *Proceedings of IJCAI-89, 10th International Joint Conference on Artificial Intelligence*, pages 1146–1152, Detroit, US, 1989.
- [17] L. Euler. *Lettres à une Princesse d’Allemagne. l’Academie Imperiale des Sciences*, 1768.
- [18] R. Fagin, A. Mendeizson, and J. Ullman. A simplified universal relation assumption and its properties. *ACM Transactions on Database Systems*, 7(3):343–360, September 1982.
- [19] Michelangelo Grigni, Dimitris Papadias, and Christos H. Papadimitriou. Topological inference. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal*, pages 901–905, 1995.
- [20] E. M. Hammer. *Logic and Visual Information*. CSLI Publications, Stanford, California, 1995.
- [21] David Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, 1988.
- [22] M. Hirakawa, M. Tanaka, and T. Ichikawa. An iconic programming system, HI-VISUAL. *IEEE Transactions on Software Engineering*, 16(10):1178–1184, October 1990.
- [23] Michael R. A. Huth and Mark D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000.
- [24] S. D. Johnson, J. Barwise, and G. Allwein. Towards the rigorous use of diagrams in reasoning about hardware. In Gerald Allwein and Jon Barwise, editors, *Logical Reasoning with Diagrams*, pages 201–223. Oxford University Press, 1996.
- [25] O. Lemon. Comparing the Efficacy of Visual Languages. In Dave Barker-Plummer, David I. Beaver, Johan van Benthem, and Patrick Scotto di Luzio, editors, *Words, Proofs, and Diagrams*, pages 47–69. CSLI Publications, Stanford, California, 2002.
- [26] O. Lemon and I. Pratt. Spatial Logic and the Complexity of Diagrammatic Reasoning. *Machine Graphics and Vision*, 6(1):89–109, January 1997.
- [27] H. J. Levesque. Making believers out of computers. In J. Mylopoulos and M. L. Brodie, editors, *Artificial Intelligence & Databases*, pages 69–82. Kaufmann Publishers, INC., San Mateo, CA, 1989.
- [28] K. Meinke and J. V. Tucker. Universal algebra. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science: Background - Mathematical Structures (Volume 1)*, pages 189–411. Clarendon Press, Oxford, 1992.
- [29] K. Myers and K. Konolige. Reasoning with analogical representations. In J. Glasgow, N. Narayanan, and N. H. Chandrasekaran, editors, *Diagrammatic Reasoning*, pages 273–301. MIT Press, Cambidge, USA, 1995.
- [30] K. L. Myers. Hybrid Reasoning Using Universal Attachment. *Artificial Intelligence*, 67:329–375, 1994.
- [31] T. Ogawa and J. Tanaka. CafePie: A Visual Programming System for CafeOBJ. In *Cafe: An Approach to Industrial Strength Algebraic Formal Methods*, pages 145–160. Elsevier Science, 2000.

- [32] L. C. Paulson. *ML for the working programmer*. Cambridge University Press, Cambridge, England, 2nd edition, 1996.
- [33] C. Peirce. *The collected papers of C. S. Peirce*. Harvard University Press, 1960.
- [34] B. C. Pierce. *Basic Category Theory for Computer Scientists*. Foundations of Computing. MIT Press, Cambridge, Massachusetts, 1991.
- [35] W. V. O. Quine. Speaking of objects. In *Ontological Relativity and Other Essays*. Columbia University Press, New York, 1969.
- [36] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
- [37] Bertrand Russell. Vagueness. *Australasian Journal of Philosophy and Psychology*, 1:84–92, 1923.
- [38] A. Shimojima. Operational constraints in diagrammatic reasoning. In Gerald Allwein and Jon Barwise, editors, *Logical Reasoning with Diagrams*, pages 27–48. Oxford University Press, 1996.
- [39] K. Stenning and O. Lemon. Aligning logical and psychological perspectives on diagrammatic reasoning. In *Thinking with Diagrams*. Kluwer, 2001.
- [40] M. Veltman. Diagrammatica: the Path to Feynman Rules. volume 4 of *Cambridge Lecture Notes in Physics*. Cambridge University Press, 1995.
- [41] W. Wechler. *Universal Algebra for Computer Scientists*. Springer-Verlag, 1992.