Graph-based Logic and Sketches

Preliminary version: Please refer others to this website instead of giving them copies of this version.

Atish Bagchi[1] and Charles Wells

September 10, 2008

[1]The first author is grateful to the Mathematical Sciences Research Institute for providing the opportunity to work on this book there. Research at MSRI was supported in part by NSF grant #DMS-9022140.

Contents

Preface

This is a preliminary version of this monograph. In certain places changes and additions to be made for the final version are marked in a paragraph beginning "To do".

This monograph presents the basic idea of forms (a generalization of Ehresmann's sketches) and their models. We also provide a formal logic that gives an intrinsically categorial definition of the concepts of assertion and proof for any particular type of form. We provide detailed examples of the machinery that enables the construction of forms, as well as examples of proofs in our formalism of certain specific assertions.

Our monograph requires familiarity with the basic notions of mathematical logic as in Chapters 2 through 5 of [Ebbinghaus *et al.*, 1984], and with category theory and sketches as in Chapters 1 through 10 of [Barr and Wells, 1999]. We specifically presuppose that finite-limit sketches and their models are known. Some notation for these ideas is established in Section 3.

This work is a combination and revision of the work in [Bagchi and Wells, 1995] and [Bagchi and Wells, 1996]. It is better because of conversations we have had with Robin Cockett and Colin McLarty. We are grateful to Frank Piessens and the referees for careful readings of earlier versions that uncovered errors, and to Max Kelly, Anders Kock and Steve Lack for supplying references. The names "string-based logic" and "graph-based logic" were suggested by Peter Freyd. The diagrams were prepared using K. Rose's `xypic`.

Atish Bagchi
Charles Wells

Chapter 1

Introduction

1.1 Brief outline

Sketches as a method of specification of mathematical structures are an alternative to the string-based specification employed in mathematical logic. They have been proposed as suitable for the specification of data types and programs [Gray, 1987], [Gray, 1989], [Wells and Barr, 1988], [Duval and Sénéchaud, 1994], and some work on implementation has been carried out [Gray, 1990], [Duval and Reynaud, 1994a], [Duval and Reynaud, 1994b].

In [Wells, 1990] the second author introduced the notion of **form**, a graph-based method of specification of mathematical structures that generalizes Ehresmann's sketches. Forms are a proper generalization of sketches: a form can have a model category that cannot be the model category of a sketch (Section 6.6). Forms were generalized to 2-categories by Power and Wells [1992] (where the word "sketch" was used instead of "form"). Other generalizations of sketches are given in [Lair, 1987], [Makkai, 1993a] and [Kinoshita *et al.*, 1997].

Sketch theory has been criticized as being lacunary when contrasted with logic because it apparently has nothing corresponding to proof theory. In this monograph, we outline a uniform proof theory for all types of sketches and forms. We show that, in the case of finite-product sketches, this results in a system with the same power as equational logic.

1.2 Types of Logic

1.2.1 String-based logic

Traditional treatments of formal logic provide:

SBL.1 A syntax for terms and formulas. The formulas are typically strings of symbols defined recursively by a production system (grammar), and the complete syntax of each term or formula is provided by the corresponding parsing tree (formation tree). To deduce the tree from the string of symbols requires fairly sophisticated pattern matching by the reader, or else a parsing mechanism in a computer program.

SBL.2 Inference relations between sets of formulas. This may be given by structural induction on the construction of the formulas, so that to verify an inference relation requires understanding the parse trees of the formulas involved.

SBL.3 Rules for assigning meaning to formulas (semantics) that are sound with respect to the inference relation. The semantics may also be given by structural induction on the construction of the formulas.

First order logic, the logic and semantics of programming languages, and the languages that have been formulated for various kinds of categories are all

commonly described by SBL 1–3.

The strings of symbols that constitute the terms and formulas are ultimately modeled on the sentences mathematicians speak and write when proving theorems. Thus first order logic in particular is a *mathematical model* of the way in which mathematicians reason. The terms and formulas are ordinary mathematical objects.

1.2.2 Graph-based logic

Mathematicians with a category-theoretic point of view frequently state and prove theorems using graphs and diagrams (described in Chapter 2). The graphs, diagrams, cones and other data of a sketch or form are formal objects that correspond to the graphs and diagrams used by such mathematicians in much the same way in which the formulas of traditional logic correspond to the sentences mathematicians use in proofs.

The functorial semantics of sketches and forms corresponds to item SBL.3 in the list in Section 1.2.1. This semantics is sound in the informal sense that it preserves by definition the structures given in the sketch or form. The analogy to the semantics of traditional model theory is close enough that sketches and forms and their models fit the definition of "institution" ([Goguen and Burstall, 1986]), which is an abstract notion of a logical system having syntactic and semantic components. This is described in detail for the case of sketches in [Barr and Wells, 1999], Section 10.3. Note that the soundness of functorial semantics appears trivial when contrasted with the inductive proofs of soundness that occur in string-based logic because the semantics functor is not defined recursively.

This monograph exhibits a structure in the theory of sketches and forms that corresponds to items SBL.1 and SBL.2. The data making up the structure we give do not correspond in any simple way to the data involved in items SBL.1 and SBL.2 of traditional logic; we discuss the relationship in Section 8.6.

1.3 Forms

Forms are parametrized by the type of constructions they allow. Let Cat be the finite-limit sketch for categories (described in detail in Section 7.2). Let E denote a finite-limit sketch whose models are a type of category with structure that is essentially algebraic over categories (this means that E contains Cat as a subsketch with the property that all the objects of E are limits of finite diagrams in Cat). Let $\mathbf{E}$ denote the finite-limit theory generated by E. E is called a **constructor space sketch** and $\mathbf{E}$ is a **constructor space**.

$\mathbf{E}$-categories are then the models of $\mathbf{E}$ in the category of sets. (This is described in detail in Chapter 6.) The kinds of categories that can be described in this way include categories with specified finite products, categories with specified limits or colimits over any particular set of diagrams, cartesian closed categories, regular categories, toposes, and many others (always with specified structure rather than structure determined only up to isomorphism — see Chapter 13). Finite limit sketches for several specific instances of $\mathbf{E}$ are given

in Chapter 7.

An $\mathbf{E}$-form F is a graph-based structure that allows the specification of any kind of construction that can be made in any $\mathbf{E}$-category. A model of F in an $\mathbf{E}$-category $\mathcal{C}$ is, informally, an instance of that construction in $\mathcal{C}$ (see the remarks in 3.4). Forms are defined precisely in Section 6.4. Ordinary sketches can also be realized as forms.

As an example, let $\mathbf{CCC}$ be a finite-limit theory for Cartesian closed categories (one is outlined in Section 7.6). It is possible to require that a certain object in a $\mathbf{CCC}$-form F be the "formal function space" A^B of two other objects A and B of the form. This means that in any model $\mathfrak{M}$ of the form in a Cartesian closed category $\mathcal{C}$, the value of $\mathfrak{M}(A^B)$ is in fact the function space $\mathfrak{M}(A)^{\mathfrak{M}(B)}$ in $\mathcal{C}$. The object A^B is not itself a function space; it is an object of a form (that is, a generalized sketch), not of a Cartesian closed category. That is why it is called a *formal* function space.

Forms have much more expressive power than sketches as originally defined by Ehresmann, in which only limits and colimits can be specified.

An $\mathbf{E}$-form F is determined by a freely adjoined global element $\mathsf{Name}[F]$ of the limit vertex of a diagram in $\mathbf{E}$, obtaining a category $\mathsf{SynCat}[\mathbf{E}, F]$. More details are in Section 6.4. An assertion in this setting is a potential factorization (PF) (defined precisely in Section 8.1) of an arrow of the $\mathsf{SynCat}[\mathbf{E}, F]$ through an arrow into its codomain. The assertion is valid if the PF does indeed factorize in every model of $\mathsf{SynCat}[\mathbf{E}, F]$.

Instead of the set of rules of deduction of a traditional theory, we have a set of rules of construction. More precisely, we give in Section 4.4 a system of construction rules that produce all the objects and arrows of the categorial theory of a finite-limits sketch. These rules apply in particular to $\mathsf{SynCat}[\mathbf{E}, F]$, which is constructed in 6.5 as such a categorial theory.

We say that the potential factorization is **deducible** if there is an actual factorization in $\mathsf{SynCat}[\mathbf{E}, F]$. Such an arrow must be constructible by the rules in Section 4.4. Thus the usual system of inference is replaced by a system of construction of arrows in the finite-limit category $\mathsf{SynCat}[\mathbf{E}, F]$ (no matter what type of category is sketched by E). This system is sound and complete with respect to models (Section 8.2).

The fact that we have assumed finite-limit sketches as given prior to the general definition of $\mathbf{E}$-form is basic to the strategy of this monograph, which is to make finite-limit logic the logic for all forms (Section 8.6) – but *not* the logic of the model category. The variation in what can be proved, for example for finite-product forms ($\mathbf{E}$-forms where $\mathbf{E} = \mathbf{FinProd}$ as in Section 7.3) as contrasted with cartesian closed forms ($\mathbf{E}$ is $\mathbf{CCC}$ as in Section 7.6) is entirely expressed by the choice of $\mathbf{E}$ and has no effect on the rules of construction.

In [Goguen and Meseguer, 1982], Goguen and Meseguer produced a sound and complete entailment system for multisorted equational logic. In Chapter 10, we verify that the theorems of that logic for a particular signature and equations all occur as actual factorizations in $\mathsf{SynCat}[\mathbf{FinProd}, F]$, where F is a $\mathbf{FinProd}$ form induced (in a manner to be described) by the given signature and equations. We also compare the expressive powers of these two systems.

[**To do:** ***The converse construction: Show how to create a multi-sorted equational theory with the same models as a*** $\mathsf{\textbf{FinProd}}$ ***form, and use the completeness theorem in Section 8.2 to show that every actual factorization of the form arises from a deduction in the theory.***]

1.4 Glossary

This monograph introduces a large number of structures with confusingly similar roles. We list the most important here with a reference to the section in which they are defined.

- $\mathsf{LinTh}[L]$, 3.1.
- $\mathsf{FinLimTh}[\mathsf{S}]$, 3.3.
- $\mathbf{\mathsf{E}}$ (= $\mathsf{FinLimTh}[E]$), 6.1.
- $\mathsf{SynCat}[\mathbf{\mathsf{E}}, F]$, 6.4.
- $\mathsf{CatTh}[\mathbf{\mathsf{E}}, F]$, 6.6.2.

Chapter 2

Preliminaries

2.1 Lists

Given a set A, $\mathsf{List}[A]$ denotes the set of lists of elements of A, including the empty list. The kth entry in a list w of elements of A is denoted by w_k and the length of w is denoted by $\mathsf{Length}[w]$. If $f : A \to B$ is a function, $\mathsf{List}[f] : \mathsf{List}[A] \to \mathsf{List}[B]$ is by definition f "mapped over" $\mathsf{List}[A]$: If w is a list of elements of A, then the kth entry of $\mathsf{List}[f](w)$ is by definition $f(w_k)$. This makes List a functor from the category of sets to itself.

2.2 Graphs

2.2.1 Definition A **graph** G is a mathematical structure consisting of a set $\mathsf{Nodes}[G]$ of **nodes** and a set $\mathsf{Arrows}[G]$ of **arrows**, with two functions $\mathsf{source} : \mathsf{Arrows}[G] \to \mathsf{Nodes}[G]$ and $\mathsf{target} : \mathsf{Arrows}[G] \to \mathsf{Nodes}[G]$.

Graphs may be pictured by drawing dots for nodes and an arrow a from a node m to a node n whenever $\mathsf{source}(a) = m$ and $\mathsf{target}(a) = n$. These are what category theorists customarily call "graphs". In the graph theory literature they would be called "directed multigraphs with loops."

The underlying graph of a category $\mathcal{C}$ is denoted by $\mathsf{UndGr}[\mathcal{C}]$. A subgraph H of a graph G is said to be **full** if every arrow $f : h_1 \to h_2$ of G between nodes of H is an arrow of H.

2.2.2 Remark Graphs are conceptually more primitive than the well-formed formulas used in string-based logic. Graphs are given by a linear sketch (see Section 3.1), essentially the simplest form of sketch, whereas wff's must be given by a context free grammar (recursive definition) which is equivalent to a finite-product sketch. (See [Barr and Wells, 1999], page 235.)

2.3 Diagrams

2.3.1 Definition Two graph homomorphisms $\delta : I \to G$ and $\delta' : I' \to G$ are said to be **equivalent** if there is a graph isomorphism $\phi : I \to I'$ such that

$$\begin{tikzcd} I \arrow[rr, "\phi"] \arrow[dr, "\delta"'] & & I' \arrow[dl, "\delta'"] \\ & G & \end{tikzcd} \tag{2.1}$$

commutes.

This relation is easily seen to be an equivalence relation on the set of graph homomorphisms into a graph G.

2.3.2 Definition A **diagram** in G is by definition an equivalence class of graph homomorphisms $\delta : I \to G$.

As is the practice when an object is defined to be an equivalence class, we will refer to a diagram by any member of the equivalence class.

2.3.3 Definition If $\delta : I \to G$ is a diagram, I is said to be a **shape graph** of the diagram, denoted by $\mathsf{ShpGr}[\delta]$, and G is said to be the **ambient space** of the diagram, denoted by $\mathsf{AmbSp}[\delta]$.

Observe that the ambient space of the diagram is determined absolutely, but the shape graph is determined only up to an isomorphism that makes Diagram (2.1) commute.

2.3.4 Definition Let I be a graph and $\mathcal{C}$ a category. To say that δ is a **diagram in** $\mathcal{C}$ means that $\delta : I \to \mathsf{UndGr}[\mathcal{C}]$ is a diagram. We write $\delta : I \to \mathcal{C}$ to denote this situation.

Note that $\mathcal{C}$ is part of the definition: there could be another category with the same underlying graph.

2.3.5 Definition A diagram $\delta : I \to \mathcal{C}$ in a category $\mathcal{C}$ **commutes** if whenever $(f_1, \ldots, f_n)$ and $(g_1, \ldots, g_m)$ are two paths in I with the same source and target, then

$$\delta(g_m) \circ \delta(g_{m-1}) \circ \cdots \circ \delta(g_1) = \delta(f_n) \circ \delta(f_{n-1}) \circ \cdots \circ \delta(f_1)$$

in $\mathcal{C}$.

Observe that commuting is defined only for diagrams in a category. More details about this may be found in [Barr and Wells, 1999], Section 4.1.5.

2.4 Convention on drawing diagrams

It is customary to draw a diagram without naming its shape graph. We adopt the following convention: If a diagram is represented by a drawing, the shape graph of the diagram is a graph that has one node for each object shown in the drawing and one arrow for each arrow shown, with source and target as shown. Two objects at different locations in the drawing correspond to two different nodes of the shape graph, *even if the objects have the same label*, and an analogous remark applies to arrows. Thus the traditional presentation of a graph, as in (2.2), reveals the equivalence class of the graph but not precisely which shape graph is used (which is irrelevant in any case).

2.4.1 Example The diagram (2.2)

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & A \arrow[d, "g"] \\ B \arrow[r, "\mathsf{Id}[B]"'] & B \end{tikzcd} \tag{2.2}$$

called δ, has shape graph

$$\begin{tikzcd} h \arrow[r, "t"] \arrow[d, "u"'] & i \arrow[d, "v"] \\ j \arrow[r, "x"'] & k \end{tikzcd} \tag{2.3}$$

so that $\delta(h) = A$, $\delta(i) = A$, $\delta(v) = g$, $\delta(x) = \mathsf{Id}[B]$ and so on. Diagram (2.4) below also has shape graph (2.3) (or one isomorphic to it, of course):

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "h"] \\ C \arrow[r, "k"'] & D \end{tikzcd} \tag{2.4}$$

On the other hand, Diagram (2.5) below

$$\begin{tikzcd} A \arrow[rr, "f"] \arrow[dr, "g"'] & & A \arrow[dl, "g"] \\ & B & \end{tikzcd} \tag{2.5}$$

is not the same diagram as (2.2). It has shape graph

$$\begin{tikzcd} i \arrow[rr, "u"] \arrow[dr, "v"'] & & j \arrow[dl, "w"] \\ & k & \end{tikzcd} \tag{2.6}$$

2.4.2 Remark The reader should observe that we use "diagram" and "commutative diagram" both in the object language and the metalanguage. For example, in Section 2.3 we refer to Diagram (2.1), which must commute in the

category of graphs. Note that we did not mention its shape graph, but according to the principles just enunciated that shape graph must be (isomorphic to)

$$\begin{tikzcd} i \arrow[rr, "u"] \arrow[dr, "v"'] & & j \arrow[dl, "w"] \\ & k & \end{tikzcd} \tag{2.7}$$

2.4.3 Remark Diagrams are customarily drawn as planar or nearly planar graphs or as perspective representations of three-dimensional graphs. A well-drawn graph reveals a lot of information quite efficiently to human beings and at the same time shows more of the structure than the formulas of traditional logic commonly do. Nevertheless, the details of the representation (nearness to planarity, symmetry when possible, and so on) that aid in human understanding are not part of the abstract structure of the diagram at all.

2.5 Cones

For any graph G and diagram $\delta : I \to G$, a cone $\Theta : v \mathrel{-\!\!\lhd} (\delta : I \to G)$ (also written $\Theta : v \mathrel{-\!\!\lhd} \delta$ if the context makes this clear) has **vertex** v denoted by $\mathsf{Vertex}[\Theta]$ and **base diagram** δ denoted by $\mathsf{BsDiag}[\Theta]$. For each node i of $\mathsf{ShpGr}[\delta]$, the formal projection of the cone Θ from $\mathsf{Vertex}[\Theta]$ to $\delta(i)$ is denoted by $\mathsf{Proj}[\Theta, i] : v \to \delta(i)$. For a category $\mathcal{C}$, a cone $\Theta : v \mathrel{-\!\!\lhd} (\delta : I \to \mathcal{C})$ is **commutative** if, for every arrow $f : i \to j$ in I, the diagram

$$\begin{tikzcd} & \mathsf{Vertex}[\Theta] \arrow[dl, "{\mathsf{Proj}[\Theta, i]}"'] \arrow[dr, "{\mathsf{Proj}[\Theta, j]}"] & \\ \delta(i) \arrow[rr, "\delta(f)"'] & & \delta(j) \end{tikzcd} \tag{2.8}$$

commutes.

In the following, we are concerned with categories with specified finite limits. In such categories, the specified limit cone of a diagram δ will be denoted by $\mathsf{LimCone}[\delta] : \mathsf{Lim}[\delta] \mathrel{-\!\!\lhd} \delta$. This specifically applies to Rule $\exists$LIM of 4.4.1.

2.6 Fonts

In general, variable objects are given in slant or script notation and specific objects (given by proper names) are given in upright notation. In more detail, we have the following notational scheme.

1. Specific data constructors, such as List, and specific fieldnames for complex objects, such as $\mathsf{Nodes}[G]$, are given in sans serif and are capitalized as shown.

2. Specific objects and arrows of sketches or forms are also given in $\mathsf{sans\ serif}$.
3. Specific constructor spaces, such as **FinLim** and **CCC**, are given in **bold sans serif**. We use **E** to denote a variable constructor space because of the unavailability of bold slanted sans serif.
4. Specific categories other than constructor spaces, such as **Set**, are given in **boldface**.
5. Diagrams (specific and variable) are named by lowercase Greek letters, such as δ and γ.
6. Cones (specific and variable) are named by uppercase Greek letters, such as Φ and Ψ.
7. Models (specific and variable) are given in uppercase fraktur, for example $\mathfrak{M}$, $\mathfrak{C}$.
8. Variable sketches and forms are given in *slanted sans serif*.
9. Variable categories other than constructor spaces are given in script, for example $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$.
10. Other variable objects are given in math italics, such as a, b, c or (especially arrows) in lowercase Greek letters.

Chapter 3

Sketches

We use a general concept of form described in Chapter 6 that is based on the concept of finite limit sketch, a particular case of projective sketch due to Ehresmann. In this section, we review briefly some aspects of linear and finite-limit sketches that are relevant later.

3.1 Linear sketches

3.1.1 Definition A **linear sketch** L is a pair $(\mathsf{Graph}[L], \mathsf{Diagrams}[L])$ where $\mathsf{Graph}[L]$ is a graph and $\mathsf{Diagrams}[L]$ is a set of finite diagrams in $\mathsf{Graph}[L]$.

3.1.2 Definition A **model** $\mathfrak{M}$ of a linear sketch S in a category $\mathcal{C}$ is a graph homomorphism $\mathfrak{M} : \mathsf{Graph}[\mathsf{S}] \to \mathsf{UndGr}[\mathcal{C}]$ that takes the diagrams in $\mathsf{Diagrams}[\mathsf{S}]$ to commutative diagrams in $\mathcal{C}$. A **morphism of models** is a natural transformation.

3.1.3 Definition The **linear theory** generated by a linear sketch L is the category obtained from the free category generated by $\mathsf{Graph}[L]$ by imposing the least congruence relation that makes the diagrams in $\mathsf{Diagrams}[L]$ commute. Here we call the linear theory $\mathsf{LinTh}[L]$.

3.1.4 Definition The **universal model** of a linear sketch L, denoted by $\mathsf{LinUnivMod}[L] : L \to \mathsf{LinTh}[L]$, is the morphism of sketches whose underlying morphism is the induced graph homomorphism (quotient map) from $\mathsf{Graph}[L]$ to the graph $\mathsf{UndGr}\Big[\mathsf{LinTh}[L]\Big]$.

3.1.5 Remark Linear sketches are so called because the underlying functor from a model in **Set** preserves products and coproducts.

3.2 Finite-limit sketches

3.2.1 Definition A **finite-limit sketch** S is a triple

$$(\mathsf{Graph}[\mathsf{S}], \mathsf{Diagrams}[\mathsf{S}], \mathsf{Cones}[\mathsf{S}])$$

where $\mathsf{Graph}[\mathsf{S}]$ is a graph, $\mathsf{Diagrams}[\mathsf{S}]$ is a set of finite diagrams in $\mathsf{Graph}[\mathsf{S}]$, and $\mathsf{Cones}[\mathsf{S}]$ is a set of cones in $\mathsf{Graph}[\mathsf{S}]$, each to a finite diagram in $\mathsf{Graph}[\mathsf{S}]$ (which need not be in $\mathsf{Diagrams}[\mathsf{S}]$).

3.2.2 Definition For finite-limit sketches S and S', a **sketch morphism** $\mathfrak{m} : \mathsf{S} \to \mathsf{S}'$ is a graph homomorphism $\mathfrak{m} : \mathsf{Graph}[\mathsf{S}] \to \mathsf{Graph}[\mathsf{S}']$ that takes the diagrams in $\mathsf{Diagrams}[\mathsf{S}]$ to diagrams in $\mathsf{Diagrams}[\mathsf{S}']$ and the cones in $\mathsf{Cones}[\mathsf{S}]$ to cones in $\mathsf{Cones}[\mathsf{S}']$.

3.2.3 Definition A **model** $\mathfrak{M}$ of a finite-limit sketch $\mathcal{S}$ in a category $\mathcal{C}$ is a graph homomorphism $\mathfrak{M}\colon \mathsf{Graph}[\mathcal{S}] \to \mathsf{UndGr}[\mathcal{C}]$ that takes the diagrams in $\mathsf{Diagrams}[\mathcal{S}]$ to commutative diagrams in $\mathcal{C}$ and the cones in $\mathsf{Cones}[\mathcal{S}]$ to limit cones in $\mathcal{C}$. A **morphism of models** is a natural transformation.

3.2.4 Definition The **forgetful functor** UndSk from the category of small categories with finite limits and finite-limit preserving functors to the category of finite-limit sketches takes a category $\mathcal{C}$ to $(\mathsf{UndGr}[\mathcal{C}], D, L)$ where D is the set of all finite commutative diagrams in $\mathcal{C}$ and L is the set of all limit cones in $\mathcal{C}$ to finite diagrams in $\mathcal{C}$.

3.3 Categorial theories of finite-limit sketches

3.3.1 Definition Let S be a finite-limit sketch. The **finite-limit theory** generated by $\mathcal{S}$, denoted by $\mathsf{FinLimTh}[\mathcal{S}]$, is a category with finite limits together with a model

$$\mathsf{FinLimUnivMod}[\mathcal{S}] : \mathcal{S} \to \mathsf{FinLimTh}[\mathcal{S}]$$

called the **universal model** of the sketch. It has the following property: For every model $\mathfrak{M}$ of $\mathcal{S}$, there is a finite-limit preserving functor $\mathsf{FinLimTh}(\mathfrak{M}) : \mathsf{FinLimTh}[\mathcal{S}] \to \mathbf{Set}$, determined uniquely up to natural isomorphism, with the property that

$$\begin{tikzcd} \mathcal{S} \arrow[r, "{\mathsf{FinLimUnivMod}[\mathcal{S}]}"] \arrow[dr, "\mathfrak{M}"'] & \mathsf{UndSk}[\mathsf{FinLimTh}[\mathcal{S}]] \arrow[d, "{\mathsf{UndSk}[\mathsf{FinLimTh}[\mathfrak{M}]]}"] \\ & \mathsf{UndSk}[\mathbf{Set}] \end{tikzcd} \tag{3.1}$$

commutes.

3.3.2 Remark It follows from the defining properties that $\mathsf{FinLimTh}[\mathcal{S}]$ is determined up to equivalence of categories and $\mathsf{FinLimUnivMod}[\mathcal{S}]$ is determined up to natural isomorphism.

3.4 Remark concerning models

Category theorists commonly use the name of a category to refer indifferently to any equivalent category. A related phenomenon occurs with respect to models, as we discuss. This monograph is concerned with syntax, and it may be necessary for clarity if not for strict correctness to distinguish between mathematical constructions that would be regarded as the "same" by many category theorists.

Let $\mathcal{Cat}$ be the finite-limit sketch for categories described in detail in Section 7.2. We may consider the following three mathematical entities.

1. Some small category $\mathcal{C}$ of one's choice.

2. A model $\mathfrak{C}$ of the sketch Cat in the category of sets which "is" or "represents" $\mathcal{C}$. This means that $\mathfrak{C}$ is a morphism of finite-limit sketches from Cat to an underlying finite-limit sketch of **Set** with the property that $\mathfrak{C}(\mathsf{ob})$ is the set of objects of $\mathcal{C}$, $\mathfrak{C}(\mathsf{ar})$ is the set of arrows of $\mathcal{C}$, $\mathfrak{C}(\mathsf{comp})$ is the composition function of $\mathcal{C}$ (up to natural isomorphism), and so on. $\mathfrak{C}$ is determined uniquely by $\mathcal{C}$ if one assumes that **Set** has specified finite limits. It is determined up to natural isomorphism in any case.

3. The model $\mathsf{FinLimTh}[\mathfrak{C}]$ of $\mathsf{FinLimTh}[\mathit{Cat}]$ induced by $\mathfrak{C}$ according to Section 3.3. This is a finite-limit-preserving functor from $\mathsf{FinLimTh}[\mathit{Cat}]$ to **Set**.

For many category theorists, $\mathcal{C}$ and $\mathfrak{C}$ denote the "same thing". Other mathematicians would disagree, saying $\mathcal{C}$ is a category (presumably for them "category" has some meaning other than "model for the sketch for categories") and $\mathfrak{C}$ is a morphism of sketches, so how could they be the same thing? This difference in point of view occurs in other situations involving models; for instance, is $(\mathbb{Z}, +)$ (the integers with addition as operation) a model of the axioms for a group, or is it correct to say that there is a model for the group axioms that "corresponds" to $(\mathbb{Z}, +)$?

The distinction between $\mathfrak{C}$ and $\mathsf{FinLimTh}[\mathfrak{C}]$ is slightly different. The first might be described as presentation data for the second. Since Lawvere, many category theorists take the view that an algebraic structure consists of the entire clone of operations rather than some generating subset of operations. From that point of view, the category $\mathcal{C}$ is the "same as" $\mathsf{FinLimTh}[\mathfrak{C}]$ rather than the same as $\mathfrak{C}$. Similarly, from the Lawverean perspective the group discussed in the previous paragraph is *determined* by saying $\mathbb{Z}$ is its underlying set and $+$ is its operation, but the group is the entire clone. (This must be distinguished from the relationship between a group and its presentation by generators and relations, although of course there is an analogy between those two situations.)

Similar remarks apply to the analogous constructions obtained when Cat is replaced by an arbitrary finite limit sketch.

In any case the following three categories are equivalent:

1. The category of small categories and functors.

2. $\mathbf{Mod}[\mathit{Cat}, \mathbf{Set}]$, the category of models of the sketch Cat in the category **Set** with natural transformations as morphisms.

3. The category of finite-limit-preserving functors from $\mathsf{FinLimTh}[\mathit{Cat}]$ to **Set** with natural transformations between them as morphisms.

In the sequel, we will distinguish between these constructions typographically as an aid to reading the monograph. We continue the discussion of this section about similar constructions in Sections 6.2, 6.6 and 6.7.

Chapter 4

Construction of finite-limit theories

4.1 Introductory comments

This chapter provides a construction of the categorial theory $\mathsf{FinLimTh}[\mathcal{S}]$ of a finite-limit sketch $\mathcal{S}$. It is essentially a special case of the construction in [Ehresmann, 1968b]. A related, more general construction is given by Duval and Reynaud in [Duval and Reynaud, 1994a].

The categorial theory $\mathsf{FinLimTh}[\mathcal{S}]$ is in fact the initial model of $\mathsf{SynCat}[\mathbf{FinLim}, \mathcal{S}]$, where $\mathbf{FinLim}$ is a finite-limit sketch defined in Section 6.4, for categories with finite limits. From that point of view, the categorial theory is a term algebra for $\mathsf{SynCat}[\mathbf{FinLim}, \mathcal{S}]$. A recursive construction of term algebras for finite-limit sketches is given in [Barr and Wells, 1999], Section 9.2.

On the other hand, for any constructor space $\mathbf{E}$ and any $\mathbf{E}$-form $\mathcal{F}$, $\mathsf{SynCat}[\mathbf{E}, \mathcal{F}]$ is (equivalent as a category to) a particular example of a finite-limit theory as described in 6.5. Thus these rules of construction may be used to construct actual factorizations (when they exist) of potential factorizations (see Section 8.1) for any $\mathbf{E}$-sketch. It is in this sense that we have reduced reasoning about arbitrary forms to reasoning (in the sense of constructing actual factorizations) about finite limits. The special character of $\mathbf{E}$-forms for a particular constructor space $\mathbf{E}$ is encoded in the constructor-space sketch that generates $\mathbf{E}$.

An approach to string-based logic that would be analogous to this setup would be a system that captured many different types of string-based logic (but not necessarily all of them) using uniform rules about manipulating the strings, with the special behavior for a particular type of logic L, for example first order logic or linear logic, encoded in a formal description of L that caused the uniform string manipulation rules to produce the correct behavior for L. As far as we know no such system has been defined. The one general approach to logic that we know about, the theory of institutions described in [Goguen and Burstall, 1986], is much more abstract than the sort of system about which we are speculating here; it does not deal with strings and rules of deduction in a computational way.

4.2 A preliminary construction

Let $\mathcal{C}$ be a category with the property that for some set (possibly empty) of diagrams in $\mathcal{C}$, limit cones have been specified. In this section, we construct a graph G and a set of diagrams D in G by $\Gamma.1$ to $\Gamma.5$ below; G and D are determined by $\mathcal{C}$. This construction is the basis of the inductive construction given in Section 4.3. The definition is deliberately made elementary (and therefore much more discursive than it might be), since it forms the basis of our deduction rules in Section 8.6.

Γ.1 If x is a node of $\mathsf{UndGr}[\mathcal{C}]$, then x is a node of G.
Γ.2 If $f : x \to y$ is an arrow of $\mathsf{UndGr}[\mathcal{C}]$, then $f : x \to y$ is an arrow of G.
Γ.3 If the composite $g \circ f$ of two arrows f and g of $\mathcal{C}$ is defined, then the diagram

$$\begin{tikzcd} \mathrm{dom}(f) \arrow[d, "f"'] \arrow[dr, "g\circ f"] & \\ \mathrm{cod}(f) \arrow[r, "g"'] & \mathrm{cod}(g) \end{tikzcd} \tag{4.1}$$

is in D.
Γ.4 If $\delta : I \to \mathcal{C}$ is a diagram in $\mathcal{C}$ that does not have a specified limit, then G contains an object $\mathsf{Lim}[\delta]$ not in $\mathcal{C}$ and a cone $\mathsf{LimCone}[\delta] : \mathsf{Lim}[\delta] \lhd \delta$, and moreover for each arrow $u : i \to j$ of I, the diagram

$$\begin{tikzcd} & \mathsf{Lim}(\delta) \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], i]}"'] \arrow[dr, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], j]}"] & \\ \delta(i) \arrow[rr, "\delta(u)"'] & & \delta(j) \end{tikzcd} \tag{4.2}$$

is in D.
Γ.5 If $\delta : I \to \mathcal{C}$ is a diagram in $\mathcal{C}$ that does not have a specified limit in $\mathcal{C}$ and Θ is a commutative cone to δ, then

1. G contains an arrow $\mathsf{Fillin}[\Theta, \delta] : \mathsf{Vertex}[\Theta] \to \mathsf{Lim}[\delta]$ not in $\mathcal{C}$.
2. For each node i of I, the diagram

$$\begin{tikzcd} \mathsf{Vertex}[\Theta] \arrow[rr, "{\mathsf{Fillin}[\Theta,\delta]}"] \arrow[dr, "{\mathsf{Proj}[\Theta, i]}"'] & & \mathsf{Lim}[\delta] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], i]}"] \\ & \delta(i) & \end{tikzcd} \tag{4.3}$$

   is in D.
3. If $k : \mathsf{Vertex}[\Theta] \to \mathsf{Lim}[\delta]$ is in $\mathcal{C}$ and, for each node i of I, the diagram

$$\begin{tikzcd} \mathsf{Vertex}[\Theta] \arrow[rr, "k"] \arrow[dr, "{\mathsf{Proj}[\Theta, i]}"'] & & \mathsf{Lim}[\delta] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], i]}"] \\ & \delta(i) & \end{tikzcd} \tag{4.4}$$

is in D, then the diagram

$$\begin{tikzcd} \mathsf{Vertex}[\Theta] \arrow[r, shift left, "{\mathsf{Fillin}[\Theta,\delta]}"] \arrow[r, shift right, "k"'] & \mathsf{Lim}[\delta] \end{tikzcd} \tag{4.5}$$

is in D.

4.3 The construction

In this section, we fix an arbitrary finite-limit sketch

$$\mathsf{S} := (\mathsf{Graph}[\mathsf{S}], \mathsf{Diagrams}[\mathsf{S}], \mathsf{Cones}[\mathsf{S}])$$

In Definitions 4.3.1 through 4.3.3 below, we construct an infinite sequence

$$\mathcal{C}_0 \xrightarrow{F_0} \mathcal{C}_1 \xrightarrow{F_1} \mathcal{C}_2 \xrightarrow{F_3} \mathcal{C}_3 \ldots \tag{4.6}$$

of categories and functors generated by S.

4.3.1 Definition Let D' be the set of diagrams consisting of all the diagrams in $\mathsf{Diagrams}[\mathsf{S}]$ and all the diagrams necessary to make each cone in $\mathsf{Cones}[\mathsf{S}]$ a formally commutative cone, namely those of the form

$$\begin{tikzcd} & \mathsf{Vertex}[\Theta] \arrow[dl, "{\mathsf{Proj}[\Theta,i]}"'] \arrow[dr, "{\mathsf{Proj}[\Theta,j]}"] & \\ \delta(i) \arrow[rr, "\delta(u)"'] & & \delta(j) \end{tikzcd} \tag{4.7}$$

for each cone $\Theta : v \lhd \delta$ in $\mathsf{Cones}[\mathsf{S}]$ and each arrow $u : i \to j$ of the shape graph of the base diagram δ of Θ. Then $\mathcal{C}_0$ is defined to be the induced category $\mathsf{LinTh}[G, D']$ as in Section 3.1.

4.3.2 Definition Let $\mathsf{S}' := (\mathsf{Graph}[\mathsf{S}'], \mathsf{Diagrams}[\mathsf{S}'])$ be the linear sketch generated by $\mathcal{C}_0$ as described in Section 4.2. Let D'' be the set of diagrams in $\mathsf{Graph}[\mathsf{S}']$ containing all the diagrams in $\mathsf{Diagrams}[\mathsf{S}']$ and, for each cone $\Theta : v \lhd \delta$ in $\mathsf{Cones}[\mathsf{S}]$ and each arrow $k : \mathsf{Vertex}[\Theta] \to \mathsf{Lim}[\delta]$, the diagram

$$\begin{tikzcd} \mathsf{Vertex}[\Theta] \arrow[r, shift left, "{\mathsf{Fillin}[\Theta,\delta]}"] \arrow[r, shift right, "k"'] & \mathsf{Lim}[\delta] \end{tikzcd} \tag{4.8}$$

Then $\mathcal{C}_1$ is defined to be the induced category $\mathsf{LinTh}[\mathsf{Graph}[\mathsf{S}], D'']$ and the image of each cone in $\mathsf{Cones}[\mathsf{S}]$ is defined to be a specified limit (it is easy to see that it is indeed a limit cone).

4.3.3 Definition Assume $\mathcal{C}_k$ has been defined for $k \geq 1$. Then $\mathcal{C}_{k+1}$ is defined to be $\mathsf{LinTh}[G, D']$ where G and D are defined from $\mathcal{C}_k$ as in Section 4.2, and $F_i : \mathcal{C}_k \to \mathcal{C}_{k+1}$ is defined to be the functor that takes each object and arrow of $\mathcal{C}$ to its congruence class in $\mathcal{C}_{k+1}$. The specified limits in $\mathcal{C}_{k+1}$ are the images of all those in $\mathcal{C}_k$ plus all those constructed by rule Γ.4.

4.3.4 Theorem *The colimit* T *of the sequence of categories and functors defined by Definitions 4.3.1 through 4.3.3 is a category with finite limits. The induced graph morphism from* G *to* $\mathsf{UndGr}[\mathsf{T}]$ *is a universal model of* S*.*

It follows from the theorem that T is the finite-limit theory of the sketch S. We denote it by $\mathsf{FinLimTh}[\mathsf{S}]$. $\mathsf{FinLimTh}[\mathsf{S}]$ is equivalent as a category to $\mathsf{CatTh}[\mathbf{\mathsf{FinLim}}, \mathsf{S}]$ (Section 6.4).

Proof Routine using the definition of colimit.□

4.3.5 Lemma (Piessens) *Every object of* $\mathsf{FinLimTh}[\mathsf{S}]$ *is a limit of a diagram built from the objects and arrows of* S*.*

(See the remarks in 6.5.1.)

Proof (Private communication from Frank Piessens.) Via the Yoneda embedding, $\mathsf{FinLimTh}[\mathsf{S}]^{\mathrm{op}}$ is a full subcategory of the category $\mathsf{Func}(\mathcal{F}, \mathbf{Set})$, where $\mathcal{F}$ the free category generated by $\mathsf{Graph}[\mathsf{S}]$. Every functor from $\mathcal{F}$ to $\mathbf{Set}$ is a colimit of representables and, hence, in $\mathsf{FinLimTh}[\mathsf{S}]$, every object is a limit of them. Thus every object of the theory is a limit of a diagram built from the objects and arrows of S.□

4.4 Rules of construction

These rules construct the objects, arrows and commutative diagrams of the category $\mathsf{FinLimTh}[\mathsf{S}]$ for a given finite-limit sketch S. The rules are given in two lists in 4.4.1 and 4.4.5 below.

4.4.1 Rules that construct objects and arrows

This list gives all the rules that construct objects and arrows in the category $\mathsf{FinLimTh}[\mathsf{S}]$. The following definition forces the distinguished cones of the sketch S to become limit cones in the theory.

4.4.2 Definition Let $\Theta : v \lhd\!\!- \delta$ be a distinguished cone of the sketch S. We define $\mathsf{LimCone}[\delta] := \Theta$ and $\mathsf{Lim}[\delta] := v$.

∃OB $\quad \dfrac{\qquad}{c} \quad$ for every object c of S.

∃ARR $\quad \dfrac{\qquad\qquad}{a \xrightarrow{f} b} \quad$ for every arrow $f : a \to b$ of S.

$$\exists\text{COMP}\quad \frac{a \xrightarrow{f} b \xrightarrow{g} c}{a \xrightarrow{g\circ f} c}$$
for every object b and pair of arrows $f : a \to b$ and $g : b \to c$ of $\mathsf{FinLimTh}[S]$.

$$\exists\text{ID}\quad \frac{c}{c \xrightarrow{\mathsf{Id}[c]} c}$$
for every object c of $\mathsf{FinLimTh}[S]$.

$$\exists\text{LIM}\quad \frac{\delta : I \to \mathsf{FinLimTh}[S]}{\mathsf{LimCone}[\delta] : \mathsf{Lim}[\delta] \mathrel{-\!\!\!\lhd} \delta}$$
for every diagram $\delta : I \to \mathsf{FinLimTh}[S]$ that is not the base of a distinguished cone of S.

$$\exists\text{FIA}\quad \frac{\Theta : v \mathrel{-\!\!\!\lhd} \delta}{\mathsf{Fillin}[\Theta, \delta] : v \to \mathsf{Lim}[\delta]}$$
for every diagram δ and every cone $\Theta : v \mathrel{-\!\!\!\lhd} \delta$ in $\mathsf{FinLimTh}[S]$.

4.4.3 Remark The first two rules are justified by the inclusion of the sketch S into $\mathsf{FinLimTh}[S]$. In rule $\exists$LIM, $\mathsf{LimCone}[\delta]$ is the specified limit of δ. The exception in rule $\exists$LIM will force the distinguished cones of S to become limit cones in $\mathsf{FinLimTh}[S]$: because of Definition 4.4.2, rule $\exists$FIA applies to those distinguished cones as well as to all cones constructed by $\exists$LIM. This remark also applies to rules CFIA and !FIA in 4.4.5.

The rules $\exists$COMP corresponds to the arrow comp and $\exists$ID to unit of the sketch for categories 7.2. $\exists$LIM and $\exists$FIA correspond to arrows in **FinLim** but not specifically to arrows of the sketch 7.5, because an arbitrary finite limit is constructed from a combination of products and equalizers.

4.4.4 Remark The rules just given construct specific objects and arrows in $\mathsf{FinLimTh}[S]$. Rule $\exists$LIM, for example, constructs a specific limit cone called $\mathsf{LimCone}[\delta]$, thus providing specified limits for $\mathsf{FinLimTh}[S]$. It is true that there are other limit cones in general for a given diagram δ, but $\mathsf{LimCone}[\delta]$ is a specific one.

Of course, in many cases, the entity constructed is the *unique* entity satisfying some property. For example, the arrow $g\circ f$ constructed by $\exists$COMP is (by definition of commutative diagram) the only one making the bottom diagram in COMPDIAG commute in $\mathsf{FinLimTh}[S]$. The arrow constructed by $\exists$ID is (by an easy theorem of category theory) the only one making the bottom diagrams in IDL and IDR commute. The arrow constructed by $\exists$FIA is (because of !FIA) the only one making the bottom diagram in CFIA commute. In connection with the point that each rule constructs a specific arrow, these observations are red herrings: in fact, each rule constructs a specific arrow with the name given, independently of any uniqueness properties arising from any other source. This point of view is contrary to the spirit of category theory. We follow it here because we are constructing syntax with an eye toward implementation in a computer language. This situation is analogous to the way in which math-

ematicians give invariant (basis-free) proofs concerning linear spaces but use bases for calculation.

Implementing the specific constructions defined above would be relatively straightforward using a modern object-oriented language. Note that we are not asserting that it would be straightforward to find a confluent and normalizing form of these rules for automatic theorem proving, only that there are no obvious difficulties in implementing them so that they could be applied in an *ad-hoc* manner.

4.4.5 Rules that construct formally commutative diagrams

The following rules produce the existence of diagrams that must commute in $\mathsf{FinLimTh}[\mathsf{S}]$.

REF

$$\frac{a \xrightarrow{f} b}{\begin{tikzcd} a \arrow[r, shift left, "f"] \arrow[r, shift right, "f"'] & b \end{tikzcd}}$$

for every arrow $f : a \to b$ of $\mathsf{FinLimTh}[\mathsf{S}]$

TRANS

$$\frac{\begin{tikzcd} a \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & b \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, shift left, "g"] \arrow[r, shift right, "h"'] & b \end{tikzcd}}{\begin{tikzcd} a \arrow[r, shift left, "f"] \arrow[r, shift right, "h"'] & b \end{tikzcd}}$$

for all objects a and b and all arrows $f, g, h : a \to b$ of $\mathsf{FinLimTh}[\mathsf{S}]$

∃DIAG

$$\frac{}{\mathsf{UnivMod}[\mathbf{FinLim}, \mathsf{S}] \circ \delta : I \to \mathsf{FinLimTh}[\mathsf{S}]}$$

for every diagram δ in the set D_{S} of distinguished diagrams of S.

COMPDIAG

$$\frac{\begin{tikzcd} a \arrow[r, "f"] & b \arrow[d, "g"] \\ & c \end{tikzcd}}{\begin{tikzcd} a \arrow[r, "f"] \arrow[dr, "g \circ f"'] & b \arrow[d, "g"] \\ & c \end{tikzcd}}$$

for every pair of arrows $f : a \to b$ and $g : b \to c$ of $\mathsf{FinLimTh}[\mathsf{S}]$.

IDL
$$\frac{\begin{array}{c} \begin{tikzcd} c \arrow[loop left, "\mathsf{Id}[c]"] \end{tikzcd} \\ b \xrightarrow{g} c \end{array}}{\begin{tikzcd} b \arrow[r, "g"] \arrow[dr, "g"'] & c \arrow[d, "\mathsf{Id}[c]"] \\ & c \end{tikzcd}}$$
for every object c and every arrow $g : b \to c$ of $\mathsf{FinLimTh}[S]$.

IDR
$$\frac{\begin{array}{c} \begin{tikzcd} c \arrow[loop left, "\mathsf{Id}[c]"] \end{tikzcd} \\ c \xrightarrow{f} d \end{array}}{\begin{tikzcd} c \arrow[r, "\mathsf{Id}[c]"] \arrow[dr, "f"'] & c \arrow[d, "f"] \\ & d \end{tikzcd}}$$
for every object c and every arrow $f : c \to d$ of $\mathsf{FinLimTh}[S]$.

ASSOC
$$\frac{a \xrightarrow{f} b \xrightarrow{g} c \xrightarrow{h} d}{\begin{tikzcd} a \arrow[r, "f"] \arrow[d, "g \circ f"'] & b \arrow[d, "h \circ g"] \arrow[dl, "g"] \\ c \arrow[r, "h"'] & d \end{tikzcd}}$$
for all arrows $f : a \to b$, $g : b \to c$ and $h : c \to d$ of $\mathsf{FinLimTh}[S]$.

CFIA
$$\frac{\begin{array}{c} i \in \mathsf{Nodes}[I] \\ \Theta : v \mathrel{-}\!\!\!\lhd \delta \end{array}}{\begin{tikzcd} v \arrow[rr, "\mathsf{Fillin}[\Theta, \delta]"] \arrow[dr, "{\mathsf{Proj}[\Theta, i]}"'] & & \mathsf{Lim}[\delta] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], i]}"] \\ & \delta(i) & \end{tikzcd}}$$

for every diagram $\delta : I \to \mathsf{FinLimTh}[S]$, every node i of I, and every cone Θ with base diagram δ.

$$\delta : I \to \mathsf{FinLimTh}[S]$$
$$\Theta : v \mathrel{-\!\!\lhd} \delta$$
$$h : v \to \mathsf{Lim}[\delta]$$
$$k : v \to \mathsf{Lim}[\delta]$$

and each of the following diagrams for each node i of I:

!FIA

$$\begin{tikzcd} v \arrow[rr, "h"] \arrow[dr, "{\mathsf{Proj}[\Theta, i]}"'] & & \mathsf{Lim}[\delta] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], i]}"] \\ & \delta(i) & \end{tikzcd}$$

$$\begin{tikzcd} v \arrow[rr, "k"] \arrow[dr, "{\mathsf{Proj}[\Theta, i]}"'] & & \mathsf{Lim}[\delta] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], i]}"] \\ & \delta(i) & \end{tikzcd}$$

$$\begin{tikzcd} v \arrow[r, shift left, "h"] \arrow[r, shift right, "k"'] & \mathsf{Lim}[\delta] \end{tikzcd}$$

for every diagram $\delta : I \to \mathsf{FinLimTh}[S]$, every cone Θ in $\mathsf{FinLimTh}[S]$ with base diagram δ, and every pair of arrows $h, k : v \to \mathsf{Lim}[\delta]$.

4.4.6 Remark Note that we do not need a rule of the form

$$\text{SYM} \quad \frac{\begin{tikzcd} a \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & b \end{tikzcd}}{\begin{tikzcd} a \arrow[r, shift left, "g"] \arrow[r, shift right, "f"'] & b \end{tikzcd}}$$

since the two diagrams exhibited are actually the same diagram (see 2.3).

4.5 A specific choice of $\mathsf{FinLimTh}[S]$

In this monograph, for a given finite-limit sketch S, we assume given a particular instance of $\mathsf{FinLimTh}[S]$: that constructed in this chapter. It has the following properties (which are not preserved by equivalence of categories):

T.1 $\mathsf{FinLimTh}[S]$ is a category with specified finite limits. (The construction explicitly produces the specified limits.)

T.2 Every arrow of $\mathsf{FinLimTh}[S]$ is a composite of projections from specified limits, fill-in arrows and arrows of the form $\mathsf{FinLimUnivMod}[S](f)$ for arrows f of the graph of S.

The following proposition is significant for this monograph when $\mathsf{FinLimTh}[S]$ is taken to be $\mathsf{SynCat}[\mathbf{E}, F]$ (defined as a particular finite-limit theory in 6.4.2 below), where F is a form.

4.5.1 Proposition *For a given sketch S, every object and every arrow of $\mathsf{FinLimTh}[S]$ is constructible by repeated applications of the constructions of Section 4.4 to the objects and arrows of the sketch S.*

Proof This proof depends on the specific choice of $\mathsf{FinLimTh}[S]$ defined in 4.5. It is clearly closed under all the constructions of Section 4.4. The properties listed in 4.5 imply that $\mathsf{FinLimTh}[S]$ is minimal with respect to the constructions of that section, so that in fact those constructions can be taken as an recursive definition of $\mathsf{FinLimTh}[S]$.$\square$

4.5.2 Remark It is also true by Lemma 4.3.5 that every object of $\mathsf{FinLimTh}[S]$ is the limit (not necessarily specified) of a diagram of the form $\mathsf{FinLimUnivMod}[S]\circ\delta$ where δ is a diagram in the graph of S. This latter property, of course, is preserved by equivalences of categories that commute with the universal model.

4.5.3 Notation It is clear that $\mathsf{FinLimTh}$ is a functor from the category of finite-limit sketches and sketch morphisms to the category of finite-limit categories and finite-limit preserving functors. For any finite-limit sketch S and morphism $\eta : S \to T$ of sketches, the induced functor between categorial theories will be denoted by

$$\mathsf{FinLimTh}[\eta] : \mathsf{FinLimTh}[S] \to \mathsf{FinLimTh}[T]$$

Chapter 5

Limits of diagrams

In this section, we develop some techniques for dealing with limits of diagrams that are used extensively in the example proofs in Sections 8.3 and 8.4.

5.1 Morphisms of Diagrams

5.1.1 Definition A **morphism of diagrams** $(\psi, \alpha) : (\delta : I \to G) \to (\delta' : I' \to G)$ is a graph morphism $\psi : I \to I'$ together with a natural transformation $\alpha : \delta' \circ \psi \to \delta$.

5.1.2 Remark It is easy to see that this definition of morphism of diagrams is compatible with the equivalence relation that defines diagrams. It was first given by Eilenberg and Mac Lane [1945] and studied further in [Kock, 1967, Guitart, 1974, Guitart and Van den Bril, 1977]. It is not the same notion of morphism of diagrams as that of [Barr, 1971], page 52, studied in [Tholen and Tozzi, 1989].

5.1.3 Definition A **strict morphism of diagrams** $\psi : (\delta : I \to G) \to (\delta' : I' \to G)$ is a graph morphism $\psi : I \to I'$ for which the diagram

$$\begin{tikzcd} I \arrow[rr, "\psi"] \arrow[dr, "\delta"'] & & I' \arrow[dl, "\delta'"] \\ & G & \end{tikzcd} \tag{5.1}$$

commutes.

5.1.4 Remark A strict morphism of diagrams is a special case of morphism of diagrams (set α to be the inclusion). All applications in this monograph use strict morphisms only.

5.1.5 Proposition *Let $(\psi, \alpha) : (\delta : I \to \mathcal{C}) \to (\delta' : I' \to \mathcal{C})$ be a morphism of diagrams in a category $\mathcal{C}$. Given a commutative cone*

$$\Theta' : \mathsf{Vertex}[\Theta'] \lhd \left(\delta' : I' \to \mathcal{C}\right)$$

there is a commutative cone

$$\Theta : \mathsf{Vertex}[\Theta] \lhd \left(\delta : I \to \mathcal{C}\right)$$

with the following properties:

a) $\mathsf{Vertex}[\Theta] = \mathsf{Vertex}[\Theta']$

b) For every $i \in \mathsf{Nodes}[I]$, $\mathsf{Proj}[\Theta, i] = \alpha i \circ \mathsf{Proj}[\Theta', \psi i]$.

Proof That Θ is commutative follows from the fact that $\mathsf{Vertex}[\Theta] = \mathsf{Vertex}[\Theta']$ and the fact that for every $f{:}i \to j$ in I, the following diagram commutes because Θ' is commutative and α is natural.

$$\begin{tikzcd} & \delta'\psi i \arrow[r, "\alpha i"] \arrow[d, "\delta'\psi f"] & \delta i \arrow[d, "\delta f"] \\ \mathsf{Vertex}[\Theta'] \arrow[ur, "{\mathsf{Proj}[\Theta', \psi i]}"] \arrow[dr, "{\mathsf{Proj}[\Theta', \psi j]}"'] & & \\ & \delta'\psi j \arrow[r, "\alpha j"] & \delta j \end{tikzcd} \tag{5.2}$$

5.1.6 Corollary *Let $(\psi, \alpha) : (\delta : I \to G) \to (\delta' : I' \to G)$ be a morphism of diagrams. Then there is a unique arrow $\phi : \mathsf{Lim}[\delta'] \to \mathsf{Lim}[\delta]$ for which for all nodes i of I,*

$$\begin{tikzcd} \mathsf{Lim}[\delta'] \arrow[r, "\phi"] \arrow[d, "{\mathsf{Proj}[\mathsf{Lim}[\delta'], \psi i]}"'] & \mathsf{Lim}[\delta] \arrow[d, "{\mathsf{Proj}[\mathsf{Lim}[\delta], i]}"] \\ \delta'\psi i \arrow[r, "\alpha i"'] & \delta i \end{tikzcd} \tag{5.3}$$

Proof This follows from Proposition 5.1.5 by letting $\Theta' := \mathsf{LimCone}[\delta']$ and then setting ϕ to be the fill-in arrow from Θ to $\mathsf{Lim}[\delta]$, where Θ is the cone defined in Proposition 5.1.5. □

5.1.7 Remark When the target of the diagrams is a category $\mathcal{C}$ with finite limits, the preceding constructions make Lim a contravariant functor from the category of diagrams to $\mathcal{C}$.

5.2 Restrictions of diagrams

5.2.1 Definition Let $\delta : I \to \mathcal{C}$ be a diagram and $\mathsf{Incl}[J \subseteq I] : J \to I$ an inclusion of graphs. The **restriction** of δ to J, denoted by $\delta|_J$, is $\delta \circ \mathsf{Incl}[J \subseteq I] : J \to \mathcal{C}$. $\delta|_J$ is called a **subdiagram** of δ.

5.2.2 Remark $\mathsf{Incl}[J \subseteq I]$ is a strict morphism of diagrams from $\delta|_J$ to δ.

5.2.3 Definition Let $\delta : I \to \mathcal{C}$ be a diagram and $\mathsf{Incl}[J \subseteq I] : J \to I$ an inclusion of graphs. Let $\Theta : v \lhd \delta$ be a cone. The **base-restriction of Θ to J** is defined to be the cone $\Theta|^J : v \lhd (\delta \circ \mathsf{Incl}[J \subseteq I])$ with vertex v and projections defined by $\mathsf{Proj}[\Theta|^J, j] := \mathsf{Proj}[\Theta, j] : v \to \delta(j)$ for all nodes j of J. In this case, we also say that Θ is a **base-extension** of $\Theta|^J$.

5.2.4 Remark If Θ is commutative, then so is $\Theta|^J$.

5.2.5 Definition Let $\delta : I \to \mathcal{C}$ be a diagram and J a subgraph of I. Then the subdiagram $\delta|_J$ is said to **dominate** δ, or to be **dominant in** δ, if every commutative cone $\Theta : v \lhd (\delta|_J)$ in $\mathcal{C}$ has a unique base extension to a commutative cone $\Theta' : v \lhd \delta$ with the same vertex.

5.2.6 Remark Tholen and Tozzi [1989] give a condition ("confinality") on I and J such that any diagram based on I is dominated by its restriction to J. One type of dominance that their condition does not cover is the case in which δ is obtained from $\delta|_J$ by adjoining a limit cone over a subdiagram of $\delta|_J$ (see Section 5.4.)

5.3 Limits of subdiagrams

5.3.1 Remark Some of the definitions and lemmas in this section have variants in which one has graph homomorphisms rather than inclusions. We shall not, however, need these.

5.3.2 Lemma *Let $\delta : I \to \mathcal{C}$ be a diagram and let J be a subgraph of I with inclusion $\mathsf{Incl}[J \subseteq I]$. Let $\gamma = \delta|_J$. Then there is a unique arrow $\phi : \mathsf{Lim}[\delta] \to \mathsf{Lim}[\gamma]$ such that for all nodes j of J,*

$$\begin{tikzcd}
\mathsf{Lim}[\delta] \arrow[rr, "\phi"] \arrow[dr, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], j]}"'] & & \mathsf{Lim}[\gamma] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], j]}"] \\
& \delta(j) &
\end{tikzcd} \tag{5.4}$$

Furthermore, if J dominates I, then ϕ is an isomorphism.

Proof The existence and uniqueness of ϕ is a special case of Corollary 5.1.6.

Now assume that γ dominates δ. Let $\Psi : \mathsf{Lim}[\gamma] \lhd \delta$ be the unique extension of $\mathsf{LimCone}[\gamma]$ to δ. Using $\exists$FIA of Section 4.4, we define

$$\psi := \mathsf{Fillin}[\Psi, \delta] : \mathsf{Lim}[\gamma] \to \mathsf{Lim}[\delta]$$

It follows from !FIA of Section 4.4 that ψ is the only arrow from $\mathsf{Lim}[\gamma]$ to $\mathsf{Lim}[\delta]$ that makes all diagrams of the form

$$\begin{tikzcd}
\mathsf{Lim}[\gamma] \arrow[rr, "\psi"] \arrow[dr, "{\mathsf{Proj}[\psi], j]}"'] & & \mathsf{Lim}[\delta] \arrow[dl, "{\mathsf{Proj}[\mathsf{LimCone}[\delta], j]}"] \\
& \gamma(j) &
\end{tikzcd} \tag{5.5}$$

commute for each node j of J. Since $\phi \circ \psi : \mathsf{Lim}[\gamma] \to \mathsf{Lim}[\gamma]$ and $\mathsf{Id}[\mathsf{Lim}[\gamma]] : \mathsf{Lim}[\gamma] \to \mathsf{Lim}[\gamma]$ both commute with all the projections to nodes of J, it fol-

lows from !FIA that $\phi \circ \psi = \mathsf{Id}[\mathsf{Lim}[\gamma]]$, A similar argument shows that $\psi \circ \phi = \mathsf{Id}[\mathsf{Lim}[\delta]]$, so that ϕ is an isomorphism. □

5.4 Special cases of extending diagrams

Here we define some special cases of dominance that are easy to recognize.

5.4.1 Definition Let graphs I and J be given such that $J \subseteq I$ and I and J have the same nodes, and suppose that I has exactly one arrow $a : j \to k$ not in J. Let $\delta : I \to \mathcal{C}$ be a diagram with the property that for all nodes j' of J and all arrows $f : j \to j'$ and $g : j' \to k$,

$$\begin{tikzcd} \delta(j) \arrow[rr, "\delta(a)"] \arrow[dr, "\delta(f)"'] & & \delta(k) \\ & \delta(j') \arrow[ur, "\delta(g)"'] & \end{tikzcd} \tag{5.6}$$

commutes in $\mathcal{C}$. Then we say δ **extends** $\delta|_J$ **by adjoining a composite**.

5.4.2 Definition Let I and J be graphs with the following properties:

ACC.1 $J \subseteq I$.
ACC.2 I has exactly one node v not in J.
ACC.3 I has at least one arrow not in J.
ACC.4 Every arrow in I not in J has target v.

Suppose that $\delta : I \to \mathcal{C}$ is a diagram with the property that if $a : i \to v$, $b : j \to v$ and $f : i \to j$ are arrows of I, then

$$\begin{tikzcd} \delta(i) \arrow[rr, "\delta(a)"] \arrow[dr, "\delta(f)"'] & & \delta(v) \\ & \delta(j) \arrow[ur, "\delta(b)"'] & \end{tikzcd} \tag{5.7}$$

commutes. Then we say δ **extends** $\delta|_J$ by **adjoining a commutative cocone**.

5.4.3 Definition Let I, J and J' be graphs with $J' \subseteq J \subseteq I$, such that J' is full in J, I contains exactly one node v not in J, and for each node j of J', I contains exactly one arrow $p_j : v \to j$ and no other arrows not in J. Let $\delta : I \to \mathcal{C}$ be a diagram, and suppose further that δ extends $\delta|_J$ in such a way that $\delta(v)$ and the arrows $\delta(p_j)$ constitute a limit cone to $\delta|_{J'}$. Then we say that δ **extends** $\delta|_J$ **by adjoining a limit.**

5.4.4 Definition Let I be a graph and let $\delta : I \to \mathcal{C}$ be a diagram. Let J' be a nonempty subgraph of I and let $\Theta : v \mathrel{-\!\!\lhd} \delta|_{J'}$ and $\Psi : w \mathrel{-\!\!\lhd} \delta|_{J'}$ be commutative cones for which

a) Θ is a limit cone.

b) Each projection $\mathsf{Proj}[\Theta, i]$ and $\mathsf{Proj}[\Psi, i]$ is a composite of arrows in the image of δ (it follows that v and w are in the image of δ.)

Let $\phi : w \to v$ be the unique fill-in arrow given by the definition of limit, and suppose f is an arrow of I for which $\delta(f) = \phi$. Let J be the subdiagram of I obtained by omitting f. Then δ **extends** $\delta|_J$ **by adjoining a fill-in arrow**.

5.4.5 Lemma *Suppose that $\delta' : I \to \mathcal{C}$ extends $\delta : J \to \mathcal{C}$ by adjoining a composite, a commutative cocone, a limit or a fill-in arrow. Then*

$$\mathsf{Fillin}\,[\mathsf{LimCone}[\delta'|_J], \delta] : \mathsf{Lim}[\delta'] \to \mathsf{Lim}[\delta]$$

is an isomorphism.

Proof We will show in each case that $\delta|_J$ dominates δ.

In the case of adjoining a composite, it follows from the fact that all the diagrams (5.6) must commute that a commutative cone over $\delta|_J$ is already a commutative cone over δ.

If δ extends $\delta|_J$ by adjoining a commutative cocone, then in the notation of Definition 5.4.2 any cone $\Theta : u \mathrel{-\!\!\lhd} \delta|_J$ extends uniquely to a cone $\Theta' : u \mathrel{-\!\!\lhd} \delta$ by defining $\mathsf{Proj}[\Theta', v] := \delta(f) \circ \mathsf{Proj}[\Theta, i]$, where $f : i \to v$ is an arrow of I not in J.

If δ extends $\delta|_J$ by adjoining a limit, then in the notation of Definition 5.4.3, $\Theta : u \mathrel{-\!\!\lhd} \delta|_J$ extends uniquely to $\Theta' : u \mathrel{-\!\!\lhd} \delta$ by defining $\mathsf{Proj}[\Theta', v] := \mathsf{Fillin}\,[\Theta', \delta|_{J'}]$.

Finally, suppose δ extends $\delta|_J$ by adjoining a fill-in arrow. By repeatedly adjoining composites we can assume δ has the property that every projection arrow $\mathsf{Proj}[\Theta, i]$ and $\mathsf{Proj}[\Psi, i]$ (notation as in Definition 5.4.4) is in the image of δ. (We are using the fact that dominance is transitive, which is easy to show.) Now let $\Phi : x \mathrel{-\!\!\lhd} \delta|_J$ be a commutative cone and let $\delta(m) = v$, $\delta(n) = w$. It is necessary and sufficient to show that the diagram

$$\begin{tikzcd} & w \arrow[dd, "\delta f"] \\ x \arrow[ur, "{\mathsf{Proj}[\Phi, n]}"] \arrow[dr, "{\mathsf{Proj}[\Phi, m]}"'] & \\ & v \end{tikzcd} \tag{5.8}$$

commutes.

For every arrow $g : j \to j'$ of J', we have a diagram

$$
\begin{tikzcd}
 & w \arrow[r, "\mathsf{Proj}[\Psi, j]"] \arrow[d, "\delta f"] \arrow[rdd, "\mathsf{Proj}[\Psi, k]" near start] & \delta(j) \arrow[dd, "\delta(g)"] \\
x \arrow[ru, "\mathsf{Proj}[\Phi, n]"] \arrow[rd, "\mathsf{Proj}[\Phi, m]"'] & & \\
 & v \arrow[r, "\mathsf{Proj}[\Theta, k]"'] \arrow[ruu, "\mathsf{Proj}[\Theta, j]" near start] & \delta(k)
\end{tikzcd}
\tag{5.9}
$$

Let the cone $\Psi' : x \mathrel{-\!\!\lhd} \delta|_{J'}$ be defined by requiring that

$$\mathsf{Proj}[\Psi', j] = \mathsf{Proj}[\Psi, j] \circ \mathsf{Proj}[\Phi, n]$$

for every node j of J'. The upper right triangle of Diagram (5.9) commutes because Ψ is a commutative cone. It follows that Ψ' is a commutative cone. We now prove that both $\mathsf{Proj}[\Phi, m]$ and $\delta f \circ \mathsf{Proj}[\Phi, n]$ satisfy the requirements of $\mathsf{Fillin}[\Psi', \delta|_{J'}]$ in the notation of of Section 4.4. It will follow from rule !FIA in that section that Diagram (5.8) commutes, as required.

a) We must show that for all nodes j of J',

$$\mathsf{Proj}[\Psi, j] \circ \mathsf{Proj}[\Phi, n] = \mathsf{Proj}[\Theta, j] \circ \mathsf{Proj}[\Phi, m]$$

This follows from the fact that Φ is a commutative cone to J and $J' \subseteq J$.

b) We must show that for all nodes j of J',

$$\mathsf{Proj}[\Theta, j] \circ \delta f \circ \mathsf{Proj}[\Phi, n] = \mathsf{Proj}[\Psi, j] \circ \mathsf{Proj}[\Phi, n]$$

This follows from the fact that the upper left triangle inside the rectangle in Diagram (5.9) commutes because δf is a fill-in arrow.

□

Chapter 6

Forms

We provide here a definition of "form" (generalized sketch) based on [Wells, 1990]. Some of the terminology has been changed. A form is a generalization of the concept of sketch invented by Charles Ehresmann and described in [Bastiani and Ehresmann, 1972] or in [Barr and Wells, 1999]. The definitions below presuppose the concept of finite-limit sketch (see Section 3).

6.1 Constructor spaces

We will assume given a fixed finite-limit sketch Cat whose category of models is the category of small categories and functors. A specific such sketch is given explicitly in Section 7.2. The presentation that follows has Cat as an implicit parameter.

6.1.1 Definition A finite-limit sketch E together with a morphism $\eta : \mathit{Cat} \to E$ of sketches is called a **constructor space sketch** provided that every object in $\mathsf{FinLimTh}[E]$ is the limit of a finite diagram whose nodes are of the form $\mathsf{FinLimTh}[\eta](\mathsf{n})$, where n is a node of Cat. The morphism η is denoted by $\mathsf{CatStruc}[E] : \mathit{Cat} \to E$.

6.1.2 Remarks The notation "CatStruc" abbreviates "categorial structure".

Definition 6.1.1 is more general than Definition 4.1.2 in [Wells, 1990] in that $\mathsf{CatStruc}[E]$ need not be an inclusion. However, in all the examples in this monograph, $\mathsf{FinLimTh}[\mathsf{CatStruc}[E]]$ is injective on objects.

6.1.3 Definition A category of the form $\mathsf{FinLimTh}[E]$ for some constructor space sketch E is called a **constructor space**.

6.1.4 Notation We will normally denote the constructor space $\mathsf{FinLimTh}[E]$ by $\mathbf{E}$ (note the difference in fonts). In particular, we have the constructor space $\mathbf{Cat}$ corresponding to the constructor space sketch Cat given in 7.2. For this example, $\mathsf{CatStruc}[\mathit{Cat}] : \mathit{Cat} \to \mathit{Cat}$ is the identity functor. For the constructor spaces $\mathbf{FinProd}$, $\mathbf{FinLim}$ and $\mathbf{CCC}$ constructed in Chapter 7, the structure map is in each case inclusion.

6.1.5 Definition A model in **Set** of a constructor space $\mathbf{E}$ is called an **E-category**, and a morphism of such models is called an **E-functor** (see Section 6.2).

6.1.6 Remarks Recall that a model of $\mathbf{E}$ is a finite-limit preserving functor from $\mathbf{E}$ to **Set**, and a morphism of models is a natural transformation from one such functor to another.

Observe that CatStruc induces an underlying functor from the category of $\mathbf{E}$-categories to the category of categories.

Definitions 6.1.1, 6.1.3 and 6.1.5 are essentially the same as those in [Wells, 1990], and are a special case (where all 2-cells are identities) of the two-dimensional version given in [Power and Wells, 1992].

6.2 Further remarks concerning models

We continue the discussion about models begun in Section 3.4. Constructor spaces **FinProd** (for categories with specified finite products), **FinLim** (for categories with specified finite limits) and **CCC** (for Cartesian closed categories with specified structure) are given in Chapter 7. The remarks concerning models of *Cat* in Section 3.4 apply equally well to models of these and other constructor spaces.

For example, each model of **CCC** is a functor, but it corresponds to a certain Cartesian closed category *with specified structure* whose objects, arrows, sources and targets, composition, binary product structure and closed structure are all determined by the values (in the model under consideration) of certain nodes and arrows of the sketch **CCC**. Morphisms of models are Cartesian closed functors that preserve all this specified structure on the nose. Cartesian closed categories in the usual sense form a large category isomorphic to the category of models of **CCC**.

We will identify Cartesian closed categories with models in **Set** of **CCC** in the sequel, and similarly for other constructor spaces **E**. In particular a **FinProd** category is a category with specified finite products and a **FinLim** category is a category with certain specified finite limits.

The value in a model $\mathfrak{M}$ of an object v in a constructor space is the set of all examples of a particular construction that is possible in the **E**-category $\mathfrak{M}$. Section 6.3 gives an extended example of this. Thus each object of **E** represents a *type of construction* possible in an **E**-category; hence the name "constructor space".

6.3 Notation for diagrams in a constructor space

The object of $\mathsf{FinLimTh}[\mathit{Cat}]$ whose value in a model is the set of all not necessarily commutative diagrams of the form

$$\begin{tikzcd} A \arrow[r, "h"] \arrow[d, "f"'] & B \arrow[d, "k"] \\ C \arrow[r, "g"'] \arrow[ur, "x"] & D \end{tikzcd} \tag{6.1}$$

is the limit of the diagram

$$
\begin{tikzcd}
 & \mathsf{ar} \arrow[r, "\mathsf{target}"] \arrow[dl, "\mathsf{source}"'] & \mathsf{ob} & \mathsf{ar} \arrow[l, "\mathsf{source}"'] \arrow[dr, "\mathsf{target}"] & \\
\mathsf{ob} & & \mathsf{ar} \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"] & & \mathsf{ob} \\
 & \mathsf{ar} \arrow[r, "\mathsf{target}"'] \arrow[ul, "\mathsf{source}"] & \mathsf{ob} & \mathsf{ar} \arrow[l, "\mathsf{source}"] \arrow[ur, "\mathsf{target}"'] &
\end{tikzcd}
\tag{6.2}
$$

Observe that $\mathsf{FinLimTh}[\mathit{Cat}]$ is the constructor space for unrestricted categories, so that Diagram (6.2) (more precisely, its image under $\mathsf{CatStruc}[E]$) occurs in any constructor space $\mathbf{E}$.

We now describe this diagram in more detail and introduce some notation that makes the discussion of such diagrams easier to follow. We use the notation $D(n)$ to refer to the diagram shown herein with label (n), and $I(n)$ for its shape graph. For example, the limit of the diagram above is $\mathsf{Lim}[D(6.2)]$.

Every node of $D(6.2)$ is either the object ob (the object that becomes the set of objects in a model) or the object ar (the object that becomes the set of arrows in a model) of $\mathsf{FinLimTh}[\mathit{Cat}]$. For a model $\mathfrak{C}$ of $\mathsf{FinLimTh}[\mathit{Cat}]$ in **Set**, an element of $\mathfrak{C}(\mathsf{Lim}[D(6.2)]$ is a diagram in $\mathfrak{C}$, not necessarily commutative, of the form of Diagram (6.1).

In order to make the relation between Diagrams (6.1) and (6.2) clear, we give the shape graph of (6.2):

$$
\begin{tikzcd}
 & f \arrow[r, "t"] \arrow[dl, "s"'] & C & g \arrow[l, "s"'] \arrow[dr, "t"] & \\
A & & x \arrow[u, "s"] \arrow[d, "t"] & & D \\
 & h \arrow[r, "t"'] \arrow[ul, "s"] & B & k \arrow[l, "s"] \arrow[ur, "t"'] &
\end{tikzcd}
\tag{6.3}
$$

We have labeled the nodes of Diagram (6.3) by the objects and arrows that occur in Diagram (6.1) in such a way that the node named by an object or arrow of Diagram (6.1) will inhabit the value of that node in the model $\mathfrak{C}$. For example, the object A of $\mathfrak{C}$ is at the upper left corner of Diagram (6.1) and the projection arrow from $\mathfrak{C}(\mathsf{Lim}[D(6.2)])$ to $\mathfrak{C}(\mathsf{ob})$ determined by the node labeled

A of the shape graph (6.3) is a function from the set of diagrams in $\mathfrak{C}$ of the form of Diagram (6.1) to the set of objects of $\mathfrak{C}$ that takes a diagram to the object in its upper left corner. The arrows of Diagram (6.3) are labeled in accordance to their values in Diagram (6.2). *It is important to understand that each distinct arrow in Diagram (6.3) is a different arrow of the shape graph*, whether they have different labels or not.

We will combine diagrams such as Diagram (6.2) and their shape graphs into one graph by labeling the nodes of the diagram by superscripts naming the corresponding node of the shape graph. In the case of Diagram (6.2), doing this gives the following **annotated diagram**:

$$\begin{tikzcd}
 & \mathsf{ar}^f \arrow[r, "\mathsf{target}"] \arrow[dl, "\mathsf{source}"'] & \mathsf{ob}^C & \mathsf{ar}^g \arrow[l, "\mathsf{source}"'] \arrow[dr, "\mathsf{target}"] & \\
\mathsf{ob}^A & & \mathsf{ar}^x \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"] & & \mathsf{ob}^D \\
 & \mathsf{ar}^h \arrow[ul, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^B & \mathsf{ar}^k \arrow[l, "\mathsf{source}"] \arrow[ur, "\mathsf{target}"'] &
\end{tikzcd} \tag{6.4}$$

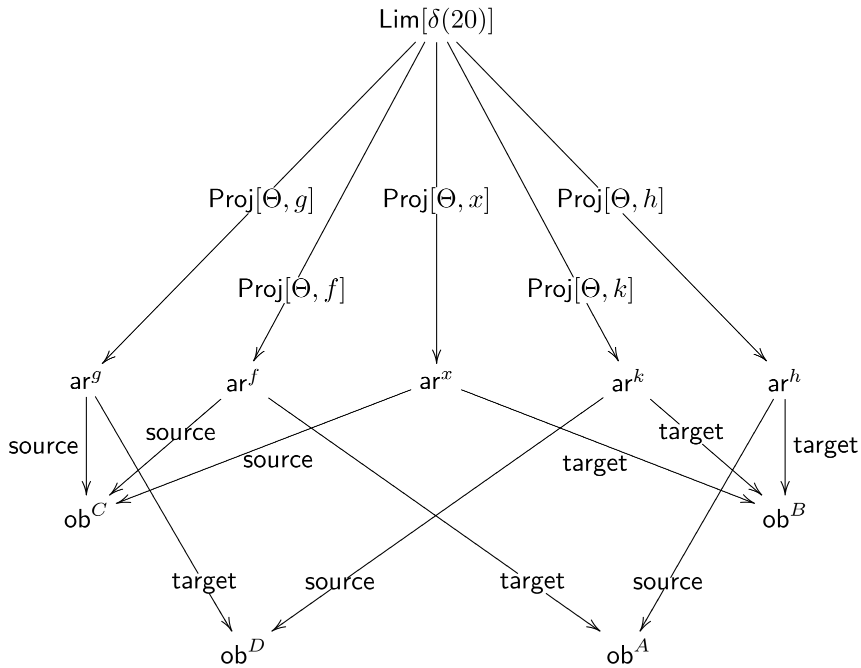
Here, the superscript A on the leftmost node indicates that the corresponding node of the shape graph is labeled A. Formally, the expression ob^A is used as the label for the node $\delta(A)$ and its use signifies that $\delta(A) = \mathsf{ob}$. That device helps the reader to see that Diagram (6.1) is indeed an element of $\mathfrak{C}(\mathsf{Lim}[D(6.2)])$.

For example, the particular arrow h of Diagram (6.1) is an element of ar, and the label ar^h in Diagram (6.4) helps one see that it is that node that projects to h in the model $\mathfrak{C}$ and that the source of h is A and that the target is B.

It is important to understand that an annotated diagram such as (6.4) denotes precisely the same diagram as (6.2). The fact that one node is labeled ob^A and another ob^B does not change the fact that *both nodes* are ob. The superscript merely gives information about the relation between Diagram (6.2) and Diagram (6.1).

Diagram (6.4) could also be drawn as the base of a limit cone Θ with limit

$\mathsf{Lim}[D(6.4)]$ (which of course is the same as $\mathsf{Lim}[D(6.2)]$) as follows.



(6.5)

Because of the typographical complexity of doing this for diagrams more complicated than Diagram (6.4), we will usually give diagrams whose limits we discuss in the form of Diagram (6.4), without showing the cone, instead of in the form of Diagram (6.5).

Showing the cone explicitly as in Diagram (6.5) nevertheless has an advantage. It makes it clear that many of the projection arrows from $\mathsf{Lim}[D(6.4)]$ are induced by others; in the particular case of Diagram (6.5), all the arrows to nodes labeled ob are induced by composing arrows to some node labeled ar with source or target. Diagram (6.4) does not make this property as easy to discover as Diagram (6.5) does.

A systematic method of translating from graphical expressions such as Diagram (6.5) to a string-based expression could presumably be based on this, following the notation introduced in [Barr and Wells, 1985], page 38. In the case of Diagram (6.5), the string-based expression would be something like this:

$$\begin{gathered}\big[\langle g,f,x,k,h\rangle \mid \mathrm{source}(g)=C,\ \mathrm{target}(g)=D,\ \mathrm{source}(f)=A,\\ \mathrm{target}(f)=C,\ \mathrm{source}(x)=C,\ \mathrm{target}(x)=B,\ \mathrm{source}(k)=B,\\ \mathrm{target}(k)=D,\ \mathrm{source}(h)=A,\ \mathrm{target}(h)=B\big]\end{gathered}$$

or in more familiar terms,

$$[\langle g,f,x,k,h\rangle \mid g:C\to D, f:A\to C, x:C\to B, k:B\to D, h:A\to B]$$

6.4 Forms

In this section, we outline those facts about forms that are needed in this monograph. More complete treatments are in [Wells, 1990] and [Power and Wells, 1992].

Let E be a constructor-space sketch, $\mathbf{E}$ (which is $\mathsf{FinLimTh}[\mathsf{E}]$) the constructor space it generates, and $\delta : I \to \mathsf{Graph}[\mathsf{E}]$ a diagram. We may freely adjoin a global element $\phi : 1 \to \mathsf{Lim}[\delta]$ to obtain a finite-limit category, denoted by $\mathbf{E}[\phi]$ in the literature and called a **polynomial category**.

6.4.1 Definition In the notation of the preceding paragraph, the **E-form** F **determined by** δ is the value $\mathfrak{I}(\phi)$ of a freely adjoined global element $\phi : 1 \to \mathsf{Lim}[\delta]$, where $\mathfrak{I}$ is the initial model of $\mathbf{E}[\phi]$ in **Set**.

6.4.2 Notation If F is an $\mathbf{E}$-form determined by δ as in the definition, we write $\mathsf{Name}[\mathsf{F}]$ for ϕ. The diagram δ is called the **description** of F. We denote $\mathbf{E}[\mathsf{Name}[\mathsf{F}]]$ by $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$ and call it the **syntactic category** of F.

6.4.3 Remark The "$\mathbf{E}$" in the notation $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$ is redundant, but helpful as a reminder of which constructor space we are using.

6.5 Constructing $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$

One way of constructing $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$ is as follows: First adjoin $\mathsf{Name}[\mathsf{F}]$ to $\mathsf{Graph}[\mathsf{E}]$ to obtain a graph G. Then define the finite-limit sketch $\mathsf{S} :=$ $(G, \mathsf{Diagrams}[\mathsf{E}], \mathsf{Cones}[\mathsf{E}])$, and finally set $\mathsf{SynCat}[\mathsf{E}, F] := \mathsf{FinLimTh}[\mathsf{S}]$. The inclusion of $\mathsf{Graph}[\mathsf{E}]$ into G is a sketch map from E to S and so generates a finite-limit preserving functor $\mathsf{Constants}[\mathsf{F}] : \mathbf{E} \to \mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$. This construction can cause considerable collapsing, for example if one adjoins a global element of an initial object.

6.5.1 Remarks A model of $\mathbf{E}$ in **Set** (a finite-limit preserving functor from $\mathbf{E}$ to **Set**) is an $\mathbf{E}$-category. A model $\mathfrak{F}$ of $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$ is an $\mathbf{E}$ category together with a chosen element of $\mathfrak{F}(\mathsf{Lim}[\delta])$, where δ is the description of F as in 6.4.2.

The functor $\mathsf{Constants}$ induces a forgetful functor (it forgets the chosen element of $\mathfrak{F}(\mathsf{Lim}[\delta])$) from set-valued (or more general) models of $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$ to models of $\mathbf{E}$.

Each object of $\mathsf{SynCat}[\mathbf{E}, \mathsf{F}]$ is the limit of a diagram in $\mathsf{E}[\mathsf{Name}[\mathsf{F}]]$ by Lemma 4.3.5, and the diagram is in some sense a description of a possible construction in any model of the form F.

6.5.2 Example As an example, consider the finite-limit sketch S with graph

$$\begin{tikzcd} A \arrow[r, shift left, "v"] & B \arrow[l, shift left, "u"] \arrow[r, "f"] & C \end{tikzcd} \tag{6.6}$$

one diagram

$$
\begin{tikzcd}
B \arrow[r, "u"] \arrow[d, "Id[B]"'] & A \arrow[dl, "v"] \\
B &
\end{tikzcd}
\tag{6.7}
$$

and one cone

$$
\begin{tikzcd}
& B \arrow[dl, "u"'] \arrow[dr, "f"] & \\
A & & C
\end{tikzcd}
\tag{6.8}
$$

This is in fact a finite-product sketch, but any such sketch is also a finite-limit sketch. Then one way to capture the information in the sketch is to take δ to be the following diagram in **FinLim** and define the **FinLim** form F determined by a freely adjoined constant $\mathsf{Name}[F] : 1 \to \mathsf{Lim}[\delta]$.

$$
\begin{tikzcd}
& & & \mathsf{ob}^A \times \mathsf{ob}^C \arrow[d, "\mathsf{prod}"] & & & \\
& \mathsf{ar}^u \arrow[dl, "\mathsf{target}"'] & & \mathsf{cone} \arrow[ll, "\mathsf{lproj}"'] \arrow[rr, "\mathsf{rproj}"] & & \mathsf{ar}^f \arrow[r, "\mathsf{target}"] \arrow[d, "\mathsf{source}"] & \mathsf{ob}^C \\
\mathsf{ob}^A & \mathsf{ar}_2 \arrow[u, "\mathsf{rfac}"] \arrow[rr, "\mathsf{comp}"] \arrow[d, "\mathsf{lfac}"] & & \mathsf{ar} & & \mathsf{ob}^B \arrow[ll, "\mathsf{unit}"'] & \\
& \mathsf{ar}^v \arrow[ul, "\mathsf{source}"] \arrow[urrrr, "\mathsf{target}"] & & & & &
\end{tikzcd}
\tag{6.9}
$$

6.6 Theories and models of forms

For completeness, we define models of forms and their morphisms, and theories of forms, using the notation of this monograph, but only briefly since these ideas are not used in this monograph. More detail and examples may be found in [Wells, 1990] or [Power and Wells, 1992].

6.6.1 Remark We are in this section identifying a model of $\mathbf{E}$ with an actual category with structure imposed by $\mathbf{E}$, and similarly for models of $\mathsf{SynCat}[\mathbf{E}, F]$. This is an example of the phenomenon mentioned in Sections 3.4 and 6.2. See also Section 6.7.

Now let F be an E-form with description $\delta : 1 \to \mathsf{E}$, so that it is named by $\mathsf{Name}[F] : 1 \to \mathsf{Lim}[\delta]$. Then $\mathsf{SynCat}[\mathsf{E}, F]$ is a finite-limit theory and so has an initial model.

6.6.2 Definition The initial model of $\mathsf{SynCat}[\mathsf{E}, F]$ is called the **E-theory** of F, denoted by $\mathsf{CatTh}[\mathsf{E}, F]$.

6.6.3 Remark The form F is an element of the value of $\mathsf{Lim}[\delta]$ in $\mathsf{CatTh}[\mathsf{E}, F]$.

6.6.4 Remark Once a finite-limit sketch S is captured as a form F as described in Example 6.5.2, it follows that $\mathsf{FinLimTh}[S]$ is naturally equivalent as a category to $\mathsf{CatTh}[\mathbf{FinLim}, F]$.

6.6.5 Definition A **model of F in an E-category** $\mathcal{C}$ is defined to be a model of $\mathsf{SynCat}[\mathsf{E}, F]$ with underlying E-category $\mathcal{C}$.

This means that $\mathcal{C}$ is the value of the functor from models of $\mathsf{SynCat}[\mathsf{E}, F]$ to models of E that forgets the element corresponding to F — see Section 6.5.1.

Let $\mathfrak{M}$ be a model of F with underlying E-category $\mathcal{C}$. Since $\mathsf{CatTh}[\mathsf{E}, F]$ is the initial model of F, there is a unique E-functor $\phi : \mathsf{CatTh}[\mathsf{E}, F] \to \mathcal{C}$ that takes $\mathsf{CatTh}[\mathsf{E}, F](\mathsf{Name}[F])$ to $\mathfrak{M}(\mathsf{Name}[F])$. In the case of familiar sketches, say finite-limit or finite-product sketches (corresponding to $\mathsf{E} = \mathbf{FinLim}$ and $\mathsf{E} = \mathbf{FinProd}$ respectively), that functor ϕ is what would usually be called the functor from the theory induced by a model of the sketch. To define for forms the entities that correspond in those cases to the actual sketch and its models involves complications and is carried out in two different ways in [Wells, 1990] and [Power and Wells, 1992].

Finally, a **morphism of models** of a form F in a category $\mathcal{C}$ is simply a natural transformation between the functors ϕ and ϕ' corresponding as described in the previous paragraph to models $\mathfrak{M}$ and $\mathfrak{M}'$ in $\mathcal{C}$.

Any work making extensive use of the entities constructed in this section will probably need to introduce more elaborate terminology and notation (for example for ϕ) than is used here. Such refinements, however, are peripheral to our concerns and we hope that this discussion is sufficiently detailed to obviate confusion.

6.7 Relationship between forms and sketches

This section continues the discussion begun in Sections 3.4 and 6.2. One constructor space is **FinLim**, defined in Section 7.5. A finite-limit sketch S in the traditional sense (a graph with diagrams and cones) corresponds to a **FinLim**-form in the construction using the methods of Example 6.5.2 and has the same models. The traditional finite-limit sketch S is an element of the value in the initial model of a certain node v of **FinLim** (not uniquely determined by S) which is the limit of a generally large and complicated diagram δ (not uniquely determined either by S or by v) in **FinLim** that specifies the graph, diagrams and cones of S.

These remarks apply to other types of Ehresmann sketches by replacing **FinLim** by the suitable constructor space.

Chapter 7

Examples of sketches for constructor spaces

Here we present constructor space sketches for certain types of categories. In each case the models are categories of the sort described and the morphisms of models are functors that preserve the structure on the nose. It is an old result that such categories can be sketched. See [Burroni, 1970a], [Burroni, 1970b], [McDonald and Stone, 1984], and [Coppey and Lair, 1988], for example.

The embedding $\mathsf{CatStruc}[\mathbf{E}]$ of Section 6.1.1 will in each case be inclusion.

7.1 Notation

We denote the ith projection in a product diagram of the form

$$\begin{tikzcd} & \mathsf{ob}^A \times \mathsf{ob}^B \arrow[dl, "p_1"'] \arrow[dr, "p_2"] & \\ \mathsf{ob}^A & & \mathsf{ob}^B \end{tikzcd}$$

as p_i, or $p_i^{A\times B}$ if the source or target is not shown. We use a similar device for the product of three copies of ob.

7.2 The sketch *Cat* for categories

This version of the sketch for categories is based on [Barr and Wells, 1999]. Another version is given in [Coppey and Lair, 1988], page 64. The first versions were done by Ehresmann [1966], [1968a] and [1968b].

7.2.1 The graph of *Cat*

The graph of the sketch for categories contains nodes as follows.

1. 1, the formal terminal object.
2. ob, the formal set of objects.
3. ar, the formal set of arrows.
4. ar_2, the formal set of composable pairs of arrows.
5. ar_3, the formal set of composable triples of arrows.

The arrows for the sketch for categories are

1. $\mathsf{unit}: \mathsf{ob} \to \mathsf{ar}$ that formally picks out the identity arrow of an object.

2. $\mathsf{source}, \mathsf{target} : \mathsf{ar} \to \mathsf{ob}$ that formally pick out the source and target of an arrow.
3. $\mathsf{comp} : \mathsf{ar}_2 \to \mathsf{ar}$ that picks out the composite of a composable pair.
4. $\mathsf{lfac}, \mathsf{rfac} : \mathsf{ar}_2 \to \mathsf{ar}$ that pick out the left and right factors in a composable pair.
5. $\mathsf{lfac}, \mathsf{mfac}, \mathsf{rfac} : \mathsf{ar}_3 \to \mathsf{ar}$ that pick out the left, middle and right factors in a composable triple of arrows.
6. $\mathsf{lass}, \mathsf{rass} : \mathsf{ar}_3 \to \mathsf{ar}_2$: lass formally takes $\langle h, g, f\rangle$ to $\langle h \circ g, f\rangle$ and rass takes it to $\langle h, g \circ f\rangle$.
7. $\mathsf{lunit}, \mathsf{runit} : \mathsf{ar} \to \mathsf{ar}_2$: lunit takes an arrow $f : A \to B$ to $\langle \mathsf{Id}[B], f\rangle$ and runit takes it to $\langle f, \mathsf{Id}[A]\rangle$.
8. Arrows $\mathsf{id} : x \to x$ as needed.

Observe that id, lfac and rfac, like p_1 and p_2, are overloaded. We will observe the same care with these arrows as with p_1 and p_2 as mentioned in Section 7.1.

7.2.2 Cones of *Cat*

ar_2 and ar_3 are defined by these cones:

$$\begin{tikzcd}
& \mathsf{ar}_2 \arrow[dl, "\mathsf{lfac}"'] \arrow[dr, "\mathsf{rfac}"] & \\
\mathsf{ar} \arrow[dr, "\mathsf{source}"'] & & \mathsf{ar} \arrow[dl, "\mathsf{target}"] \\
& \mathsf{ob} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathsf{ar}_3 \arrow[dl, "\mathsf{lfac}"'] \arrow[d, "\mathsf{mfac}"] \arrow[dr, "\mathsf{rfac}"] & \\
\mathsf{ar} \arrow[d, "\mathsf{source}"'] & \mathsf{ar} \arrow[dl, "\mathsf{target}"] \arrow[dr, "\mathsf{source}"'] & \mathsf{ar} \arrow[d, "\mathsf{target}"] \\
\mathsf{ob} & & \mathsf{ob}
\end{tikzcd}$$

7.2.3 Diagrams of *Cat*

$$\begin{tikzcd}
\mathsf{ar}_2 \arrow[d, "\mathsf{comp}"'] & \mathsf{ar}_3 \arrow[l, "\langle \mathsf{lfac}, \mathsf{mfac}\rangle"'] \arrow[d, "\mathsf{lass}"] \arrow[dr, "\mathsf{rfac}"] & \\
\mathsf{ar} & \mathsf{ar}_2 \arrow[l, "\mathsf{lfac}"] \arrow[r, "\mathsf{rfac}"'] & \mathsf{ar}
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathsf{ar}_3 \arrow[r, "\langle \mathsf{lfac}, \mathsf{mfac}\rangle"] \arrow[dl, "\mathsf{lfac}"'] \arrow[d, "\mathsf{rass}"] & \mathsf{ar}_2 \arrow[d, "\mathsf{comp}"] \\
\mathsf{ar} & \mathsf{ar}_2 \arrow[l, "\mathsf{lfac}"] \arrow[r, "\mathsf{rfac}"'] & \mathsf{ar}
\end{tikzcd} \tag{7.1}$$

$$
\begin{tikzcd}
\mathsf{ob} \arrow[d, "\mathsf{unit}"'] & \mathsf{ar} \arrow[l, "\mathsf{target}"'] \arrow[d, "\mathsf{lunit}"] \arrow[dr, "\mathsf{id}"] & \\
\mathsf{ar} & \mathsf{ar}_2 \arrow[l, "\mathsf{lfac}"] \arrow[r, "\mathsf{rfac}"'] & \mathsf{ar}
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathsf{ar} \arrow[dl, "\mathsf{id}"'] \arrow[d, "\mathsf{runit}"] \arrow[r, "\mathsf{source}"] & \mathsf{ob} \arrow[d, "\mathsf{unit}"] \\
\mathsf{ar} & \mathsf{ar}_2 \arrow[l, "\mathsf{lfac}"] \arrow[r, "\mathsf{rfac}"'] & \mathsf{ar}
\end{tikzcd}
$$

$$
\begin{tikzcd}
\mathsf{ar} \arrow[r, "\mathsf{runit}"] \arrow[dr, "\mathsf{id}"'] & \mathsf{ar}_2 \arrow[d, "\mathsf{comp}"] & \mathsf{ar} \arrow[l, "\mathsf{lunit}"'] \arrow[dl, "\mathsf{id}"] \\
& \mathsf{ar} &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{ar}_3 \arrow[r, "\mathsf{rass}"] \arrow[d, "\mathsf{lass}"'] & \mathsf{ar}_2 \arrow[d, "\mathsf{comp}"] \\
\mathsf{ar}_2 \arrow[r, "\mathsf{comp}"'] & \mathsf{ar}
\end{tikzcd}
\tag{7.2}
$$

7.3 The sketch for the constructor space **FinProd**

To get the sketch for categories with finite products, we must add the following nodes and arrows to the sketch for categories:

Nodes:

1. ta, the formal set of terminal arrows.
2. cone, the formal set of cones of the form

$$
\begin{tikzcd}
& V \arrow[dl, "p_1"'] \arrow[dr, "p_2"] & \\
A & & B
\end{tikzcd}
\tag{7.3}
$$

3. fid, the formal set of fill-in diagrams ("sawhorses") of the form

$$
\begin{tikzcd}
V \arrow[r, "h"] \arrow[d] \arrow[dr] & L \arrow[d] \arrow[dl] \\
A & B
\end{tikzcd}
\tag{7.4}
$$

where h commutes with the cone projections.

Arrows:

1. $\mathsf{ter} : 1 \to \mathsf{ob}$, that formally picks out a particular terminal object.
2. $! : \mathsf{ob} \to \mathsf{ta}$, that picks out the arrow from an object to the terminal object.
3. $\mathsf{inc} : \mathsf{ta} \to \mathsf{ar}$, the formal inclusion of the set of terminal arrows into the set of arrows.
4. $\mathsf{prod} : \mathsf{ob} \times \mathsf{ob} \to \mathsf{cone}$, that picks out the product cone over a pair of objects.
5. $\mathsf{soco} : \mathsf{fid} \to \mathsf{cone}$, that picks out the source cone of a fill-in arrow.
6. $\mathsf{taco} : \mathsf{fid} \to \mathsf{cone}$, that picks out the target cone of a fill-in arrow.
7. $\mathsf{ufid} : \mathsf{cone} \to \mathsf{fid}$, that takes a cone to the unique fill-in diagram that has the cone as source cone.
8. $\mathsf{fia} : \mathsf{fid} \to \mathsf{ar}$ that formally picks out the fill-in arrow in a fill-in diagram.

7.3.1 Cones for **FinProd**

FinProd has four cones in addition to those of the sketch for categories. One is the cone

$$1$$

over the empty diagram. The one below says that ta is the formal set of arrows to the terminal object:

$$\begin{tikzcd}
& \mathsf{ta} \arrow[dl, "\mathsf{inc}"'] \arrow[dr] & \\
\mathsf{ar} \arrow[rr, "\mathsf{target}"] \arrow[dr] & & \mathsf{ob} \\
& 1 \arrow[ur, "\mathsf{ter}"'] &
\end{tikzcd} \tag{7.5}$$

Note that in giving this cone, we are not only saying that ta is the limit of the diagram

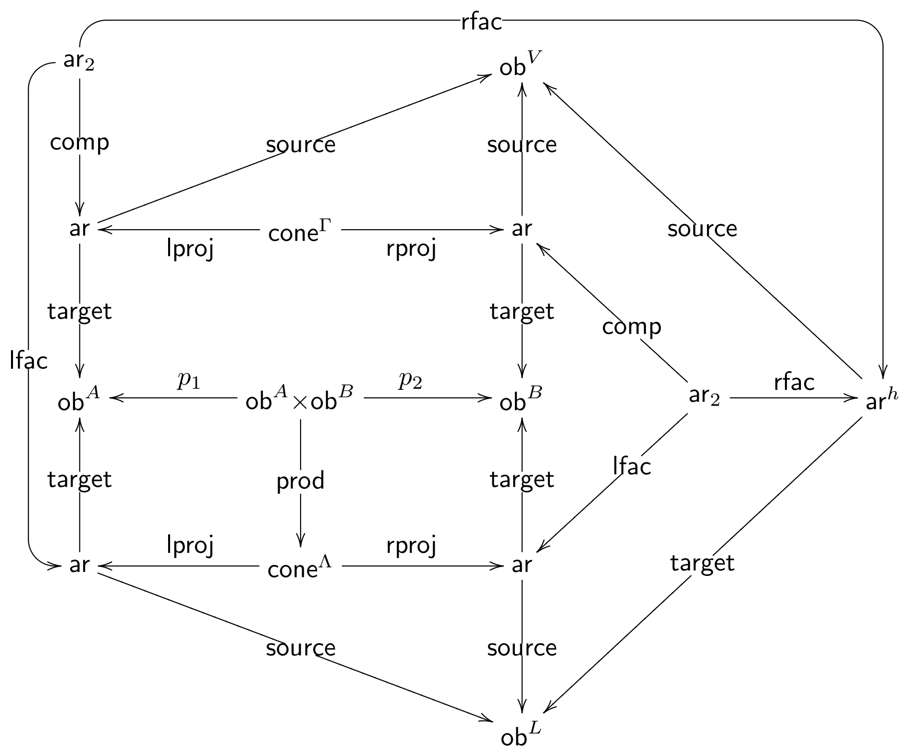
$$\begin{tikzcd}
\mathsf{ar} \arrow[rr, "\mathsf{target}"] \arrow[dr] & & \mathsf{ob} \\
& 1 \arrow[ur, "\mathsf{ter}"'] &
\end{tikzcd} \tag{7.6}$$

but also that inc is one of the projection arrows. (Indeed, this is the only projection arrow that matters, since the other two are induced.)

The following cone makes cone the formal object of cones to a discrete diagrams consisting of a pair of objects.

$$\begin{tikzcd} & \mathsf{cone} \arrow[dl, "\mathsf{lproj}"'] \arrow[dr, "\mathsf{rproj}"] & \\ \mathsf{ar} \arrow[r, "\mathsf{source}"'] & \mathsf{ob} & {} \arrow[l, "\mathsf{source}"] \end{tikzcd} \tag{7.7}$$

Finally, there must be a cone with vertex fid over Diagram 7.8 below, which is annotated to refer to Diagram (7.4), in which Γ is the cone with vertex V, Λ is the cone with vertex L and is a limit cone, and h is the fill-in arrow. In this case, the projection arrows of the cone are not shown.



(7.8)

In addition, we require:

1) The projection to cone^Γ must be soco.
2) The projection to cone^Λ must be taco.
3) The projection to ar^h must be fia.

7.3.2 Diagrams for **FinProd**

The following two diagrams make the arrow to the terminal object have the correct source and target.

$$
\begin{tikzcd}
\mathsf{ob} \arrow[r, "!"] \arrow[drr, "\mathsf{id}"'] & \mathsf{ta} \arrow[r, "\mathsf{inc}"] & \mathsf{ar} \arrow[d, "\mathsf{source}"] \\
 & & \mathsf{ob}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{ta} \arrow[r, "\mathsf{inc}"] \arrow[drr, "\mathsf{id}"'] & \mathsf{ar} \arrow[r, "\mathsf{source}"] & \mathsf{ob} \arrow[d, "!"] \\
 & & \mathsf{ta}
\end{tikzcd}
\tag{7.9}
$$

The diagram below makes the fill-in arrow to a product unique.

$$
\begin{tikzcd}
\mathsf{cone} \arrow[r, shift left, "\mathsf{ufid}"] & \mathsf{fid} \arrow[l, shift left, "\mathsf{soco}"]
\end{tikzcd}
\tag{7.10}
$$

The diagram below forces the product cone projections to have the correct targets.

$$
\begin{tikzcd}
 & & \mathsf{ob} \times \mathsf{ob} \arrow[dll, "p_1"'] \arrow[d, "\mathsf{prod}"] \arrow[drr, "p_2"] & & \\
\mathsf{ob} & \mathsf{ar} \arrow[l, "\mathsf{target}"] & \mathsf{cone} \arrow[l, "\mathsf{lproj}"] \arrow[r, "\mathsf{rproj}"'] & \mathsf{ar} \arrow[r, "\mathsf{target}"'] & \mathsf{ob}
\end{tikzcd}
\tag{7.11}
$$

7.4 Modules

As we proceed to sketch more complicated constructions, we will need to use some device to communicate the nature of the necessary diagrams, which become too large to comprehend easily. Here we introduce the first of several **modules**: diagrams that occur frequently as subdiagrams because they are needed to force the value of a node in a model to contain certain types of constructions.

Modules are a well-understood part of programming language methodology. We believe that the concept called “module” here can be made explicit enough to become part of a programming language based on the techniques of this monograph, but that work is yet to be accomplished.

7.4.1 The module for the product of objects

Every occurrence of ob that is annotated $M \times N$ must be part of a subdiagram of the following form:

$$\begin{tikzcd}
\mathsf{ob}^M & \mathsf{ar}^{p_1^{M\times N}} \arrow[l, "\mathsf{target}"'] \arrow[dr, "\mathsf{source}"] & \\
\mathsf{ob}^M \times \mathsf{ob}^N \arrow[u, "p_1"] \arrow[d, "p_2"'] \arrow[r, "\mathsf{prod}"] & \mathsf{cone} \arrow[u, "\mathsf{lproj}"] \arrow[d, "\mathsf{rproj}"] & \mathsf{ob}^{M\times N} \\
\mathsf{ob}^N & \mathsf{ar}^{p_2^{M\times N}} \arrow[l, "\mathsf{target}"] \arrow[ur, "\mathsf{source}"'] &
\end{tikzcd} \tag{7.12}$$

Henceforth, an occurrence of ob annotated $A \times B$ (for example) will be taken to *imply* the existence of a subdiagram of the form of Diagram (7.12) with M replaced with A and N replaced with B. The subdiagram will not necessarily be shown. If this is part of a diagram δ, the diagram can be reconstructed by taking the union of the shape graph of the module (7.12) and the shape graph of the part of δ that is shown on the page, and defining the diagram based on the resulting graph as the pushout of the diagram shown and the module. This is illustrated in Diagrams (7.14) and (7.15) in the next section.

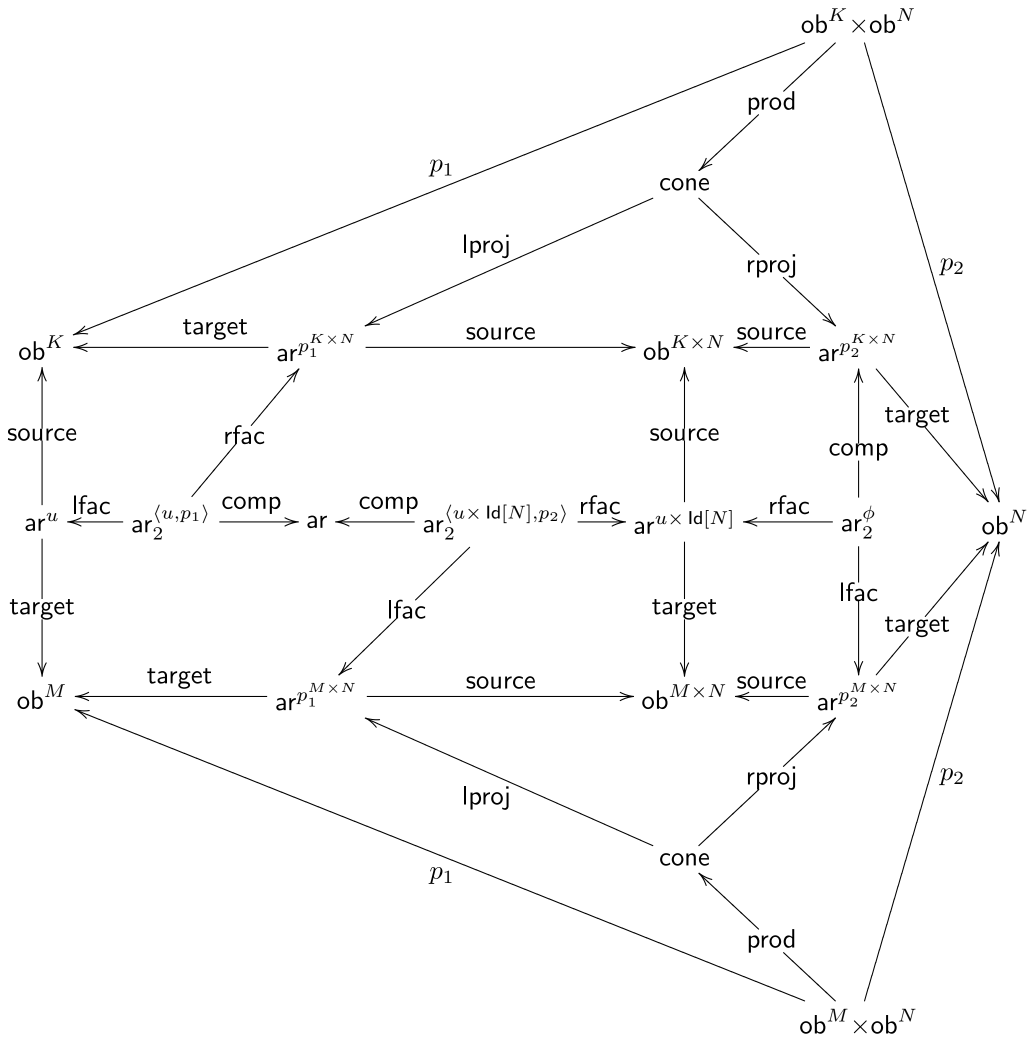
7.4.2 The module for the product of arrows

In the commutative diagram

$$\begin{tikzcd}
K \arrow[dd, "u"'] & K \times N \arrow[l, "p_1"'] \arrow[dd] \arrow[dr, "p_2"] & \\
& & N \\
M & M \times N \arrow[l, "p_2"] \arrow[ur, "p_2"'] &
\end{tikzcd} \tag{7.13}$$

the unlabeled arrow is necessarily $u \times \mathsf{Id}[N] : K \times N \to M \times N$. Such a diagram must be an element in a model of the value of Diagram (7.14) below, which is therefore a module for the product of an arrow and an identity arrow. In this

diagram, $\phi := \langle p_2^{M\times N}, u \times \mathsf{Id}[N]\rangle$.



(7.14)

More precisely, let x be an element of $\mathfrak{M}(\mathsf{Lim}[D(7.14)])$, for some category $\mathfrak{M}$ with finite limits. Then if $\mathsf{Proj}[\mathsf{Lim}[D(7.14)], h](x) = h$, then

$$\mathsf{Proj}[\mathsf{Lim}[D(7.14)], h\times \mathsf{Id}[A]](x) = h\times \mathsf{Id}[A]$$

as suggested by the notation. This will be used in Section 7.6 below.

Diagram (7.14) contains two copies of Diagram (7.12), the module for the product of two objects. The copy at the bottom is precisely Diagram (7.12), and the copy at the top is Diagram (7.12) with M replaced with K. In the sequel, a diagram such as Diagram (7.14) will be drawn without the modules,

as shown below.

$$
\begin{tikzcd}
\mathsf{ob}^K & & \mathsf{ar}^{p_1^{K\times N}} \arrow[ll, "\mathsf{target}"'] \arrow[rr, "\mathsf{source}"] & & \mathsf{ob}^{K\times N} & \mathsf{ar}^{p_2^{K\times N}} \arrow[l, "\mathsf{source}"'] \arrow[dr, "\mathsf{target}"] & \\
\mathsf{ar}^u \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"'] & \mathsf{ar}_2^{\langle u, p_1^{K\times N}\rangle} \arrow[l, "\mathsf{lfac}"'] \arrow[ur, "\mathsf{rfac}"] \arrow[r, "\mathsf{comp}"] & \mathsf{ar} & \mathsf{ar}_2^{\langle u\times \mathsf{Id}[N], p_2\rangle} \arrow[l, "\mathsf{comp}"'] \arrow[r, "\mathsf{rfac}"] \arrow[dl, "\mathsf{lfac}"] & \mathsf{ar}^{u\times \mathsf{Id}[N]} \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"'] & \mathsf{ar}_2^{\phi} \arrow[l, "\mathsf{rfac}"'] \arrow[u, "\mathsf{comp}"] \arrow[d, "\mathsf{lfac}"'] & \mathsf{ob}^N \\
\mathsf{ob}^M & & \mathsf{ar}^{p_1^{M\times N}} \arrow[ll, "\mathsf{target}"'] \arrow[rr, "\mathsf{source}"] & & \mathsf{ob}^{M\times N} & \mathsf{ar}^{p_2^{M\times N}} \arrow[l, "\mathsf{source}"'] \arrow[ur, "\mathsf{target}"'] &
\end{tikzcd}
\tag{7.15}
$$

Diagram (7.14) may be mechanically reconstructed from Diagram (7.15) and the annotations that include the symbols $M \times N$ and $K \times N$ (three of each). The shape diagram of Diagram (7.14) is the pushout of the shape diagram of Diagram (7.15) and the shape diagrams of the modules Diagram (7.12) and Diagram (7.12) with $M \leftarrow K$. Each of the latter two have four annotated nodes and six annotated arrows in common with Diagram (7.15), and the values of any two of the three smaller diagrams at a given common node or arrow is of course the same, so that Diagram (7.14) is the union of Diagram (7.15) and the two modules.

7.5 The sketch for the constructor space **FinLim**

We sketch the constructor space **FinLim** by adding data to the sketch for **FinProd** that ensure that a **FinLim**-category has equalizers of pairs of arrows. The sketch has the following nodes:

1. ppair is the formal set of parallel pairs of the form

$$
\begin{tikzcd}
A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B
\end{tikzcd}
\tag{7.16}
$$

2. econe is the formal set of diagrams

$$
\begin{tikzcd}
E \arrow[r, "u"] & A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B
\end{tikzcd}
\tag{7.17}
$$

   in which $f \circ u = g \circ u$. Of course, a cone to Diagram (7.16) also has a projection to B, but that is forced and need not be included in the data for the cone.

3. efid is the set of fill-in diagrams

$$\begin{tikzcd} & X \arrow[dl, "v"'] \arrow[d, "u"] & \\ E \arrow[r, "e"] & A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B \end{tikzcd} \tag{7.18}$$

in which $f \circ e = g \circ e$ and $u = e \circ v$.

The arrows of the sketch include:

1. $\mathsf{equ} : \mathsf{ppair} \to \mathsf{econe}$, that formally picks out the equalizer of the parallel pair.
2. $\mathsf{top}, \mathsf{bot} : \mathsf{ppair} \to \mathsf{ar}$, that pick out f and g in Diagram (7.16).
3. $\mathsf{etop}, \mathsf{ebot} : \mathsf{econe} \to \mathsf{ar}$, that pick out f and g in Diagram (7.17).
4. $\mathsf{esoco}, \mathsf{etaco} : \mathsf{efid} \to \mathsf{econe}$ that pick out the source and target cones of the fill-in arrow.
5. $\mathsf{eufid} : \mathsf{econe} \to \mathsf{efid}$ that takes a diagram of the form of Diagram (7.17) to the unique fill-in diagram that has this diagram as source cone.
6. $\mathsf{efia} : \mathsf{efid} \to \mathsf{ar}$ that picks out the fill-in arrow in a fill-in diagram.

7.5.1 Cones for FinLim

ppair is the limit of the diagram

$$\begin{tikzcd} f \arrow[r, "\mathsf{target}"] \arrow[d, "\mathsf{source}"'] & \mathsf{ob}^B \\ \mathsf{ob}^A & g \arrow[l, "\mathsf{source}"'] \arrow[u, "\mathsf{target}"'] \end{tikzcd} \tag{7.19}$$

The following projections from ppair have names: $\mathsf{soob}{:}\mathsf{ppair} \to \mathsf{ob}^A$, $\mathsf{top}{:}\mathsf{ppair} \to \mathsf{ar}^f$, and $\mathsf{bot} : \mathsf{ppair} \to \mathsf{ar}^g$.

econe is the limit of

$$\begin{tikzcd}
 & \mathsf{ar}_2 \arrow[r, "\mathsf{lfac}"] \arrow[dl, "\mathsf{comp}"'] \arrow[d, "\mathsf{rfac}"] & \mathsf{ar}^f \arrow[d, "\mathsf{source}"] \arrow[dr, "\mathsf{target}"] & \\
{} & \mathsf{ar}^e \arrow[r, "\mathsf{target}"] & \mathsf{ob}^A & \mathsf{ob}^B \\
 & \mathsf{ar}_2 \arrow[r, "\mathsf{lfac}"'] \arrow[ul, "\mathsf{comp}"] \arrow[u, "\mathsf{rfac}"] & \mathsf{ar}^g \arrow[u, "\mathsf{source}"] \arrow[ur, "\mathsf{target}"'] &
\end{tikzcd} \tag{7.20}$$

Two projections have names: $\mathsf{etop} : \mathsf{econe} \to \mathsf{ar}^f$ and $\mathsf{ebot} : \mathsf{econe} \to \mathsf{ar}^g$.

efid is the limit of the pushout of Diagram (7.20) and the following diagram. Note that the common part of the two diagrams is

$$\mathsf{ar}^e \xrightarrow{t} \mathsf{ob}^A$$

7.5.2 Remark We could have presented Diagram (7.8) as a pushout in much the same way (the common part would describe the arrow $h : V \to L$). We have deliberately varied the way we present the data in this monograph because we are not sure ourselves which approach communicates best.

$$\begin{tikzcd}
\mathsf{ar}^v \arrow[r, "\mathsf{source}"] \arrow[dd, "\mathsf{target}"'] & \mathsf{ob}^X & \mathsf{ar}^u \arrow[l, "\mathsf{source}"'] \arrow[dd, "\mathsf{target}"] \\
 & \mathsf{ar}_2 \arrow[ul, "\mathsf{rfac}"] \arrow[ur, "\mathsf{comp}"'] \arrow[d, "\mathsf{lfac}"] & \\
\mathsf{ob}^E & \mathsf{ar}^e \arrow[l, "\mathsf{source}"'] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^A
\end{tikzcd} \tag{7.21}$$

The named projections are $\mathsf{esoco}{:}\mathsf{efid} \to \mathsf{ar}^u$, $\mathsf{etaco}{:}\mathsf{efid} \to \mathsf{ar}^e$ and $\mathsf{efia}{:}\mathsf{efid} \to \mathsf{ar}^v$.

7.5.3 Diagrams for FinLim

The following diagram makes the fill-in arrow unique.

$$\begin{tikzcd}
\mathsf{econe} \arrow[r, shift left, "\mathsf{eufid}"] & \mathsf{efid} \arrow[l, shift left, "\mathsf{esoco}"]
\end{tikzcd} \tag{7.22}$$

These two diagrams ensure that the equalizer cone be a cone to the correct diagram.

$$\begin{tikzcd} \mathsf{ppair} \arrow[r, "\mathsf{equ}"] \arrow[dr, "\mathsf{top}"'] & \mathsf{econe} \arrow[d, "\mathsf{etop}"] \\ & \mathsf{ar} \end{tikzcd} \qquad \begin{tikzcd} \mathsf{ppair} \arrow[r, "\mathsf{equ}"] \arrow[dr, "\mathsf{bot}"'] & \mathsf{econe} \arrow[d, "\mathsf{ebot}"] \\ & \mathsf{ar} \end{tikzcd} \tag{7.23}$$

7.6 The sketch for the constructor space **CCC**

7.6.1 Definition A **Cartesian closed category** is a category $\mathcal{C}$ with the following structure:

CCC.1 $\mathcal{C}$ has binary products.
CCC.2 For each pair of objects A and B of $\mathcal{C}$, there is an object B^A and an arrow $\mathrm{eval}: B^A \times A \to B$.
CCC.3 For each triple of objects A, B and C of $\mathcal{C}$, there is a map

$$\lambda : \mathrm{Hom}(B \times A, C) \to \mathrm{Hom}(B, C^A) \tag{7.24}$$

such that for every arrow $f : B \times A \to C$,

$$\begin{tikzcd} B \times A \arrow[r, "\lambda f \times \mathsf{Id}[A]"] \arrow[dr, "f"'] & C^A \times A \arrow[d, "\mathrm{eval}"] \\ & C \end{tikzcd} \tag{7.25}$$

commutes.
CCC.4 For any arrow $g : B \to C^A$, $\lambda(\mathrm{eval} \circ (g \times \mathsf{Id}[A])) = g$.

Using this definition, the sketch for the constructor space for Cartesian closed categories may be built on the sketch for **FinProd** by adding the following nodes and arrows.

The nodes are:

1. twovf, the formal set of "functions of two variables", that is, arrows of the form $B \times A \to C$.

2. curry, the formal set of "curried functions" $B \to C^A$.

The sketch for **CCC** has arrows

1. $\mathsf{fs} : \mathsf{ob}^B \times \mathsf{ob}^A \to \mathsf{ob}^{B^A}$ that picks out the function space B^A of two objects B and A.

2. $\mathsf{ev} : \mathsf{ob}^B \times \mathsf{ob}^A \to \mathsf{ar}$ that picks out the arrow $\mathrm{eval}\colon B^A \times A \to B$.
3. $\mathsf{lam} : \mathsf{ar} \to \mathsf{ar}$, the formal version of the mapping λ of Diagram (7.24).
4. $\mathsf{tsource}\colon\mathsf{twovf} \to \mathsf{ob}^{B\times A}$, that picks out the source of a function $f\colon A\times B \to C$.
5. $\mathsf{ttarget}\colon\mathsf{twovf} \to \mathsf{ob}^C$, that picks out the target of a function $f\colon A\times B \to C$.
6. $\mathsf{arrow} : \mathsf{twovf} \to \mathsf{ar}^f$, that picks out the arrow f itself.
7. $\mathsf{csource} : \mathsf{curry} \to \mathsf{ob}^B$, that picks out the source of a curried function $g : B \to C^A$.
8. $\mathsf{ctarget} : \mathsf{curry} \to \mathsf{ob}^{C^A}$, that picks out the target of a curried function $f : B \to C^A$.
9. $\mathsf{arrow} : \mathsf{curry} \to \mathsf{ar}^g$, that picks out the arrow g itself.

7.6.2 Cones for **CCC**

The constructor space **CCC** must have two cones

$$\begin{tikzcd}
& \mathsf{twovf} \arrow[dl, "\mathsf{tsource}"'] \arrow[d, "\mathsf{arrow}"] \arrow[dr, "\mathsf{ttarget}"] & \\
\mathsf{ob}^{B\times A} & \mathsf{ar}^f \arrow[l, "\mathsf{source}"] \arrow[r, "\mathsf{source}"'] & \mathsf{ob}^C
\end{tikzcd} \tag{7.26}$$

$$\begin{tikzcd}
& \mathsf{curry} \arrow[dl] \arrow[d, "\mathsf{ctarget}"] \arrow[dr, "\mathsf{arrow}"] \arrow[drr, "\mathsf{csource}"] & & \\
\mathsf{ob}^C \times \mathsf{ob}^A \arrow[r, "\mathsf{fs}"'] & \mathsf{ob}^{C^A} & \mathsf{ar}^g \arrow[l, "\mathsf{target}"] \arrow[r, "\mathsf{source}"'] & \mathsf{ob}^B
\end{tikzcd} \tag{7.27}$$

7.6.3 The module for function spaces

Henceforth, we will assume the module

$$\begin{tikzcd}
\mathsf{ob}^M & \mathsf{ob}^M \times \mathsf{ob}^N \arrow[l, "p_1"'] \arrow[r, "\mathsf{fs}"] \arrow[d, "p_2"] & \mathsf{ob}^{M^N} \\
& \mathsf{ob}^N &
\end{tikzcd} \tag{7.28}$$

is attached whenever an occurrence of ob is annotated by M^N. Note that this occurred in Diagram (7.27).

7.6.4 Diagrams for CCC

Diagram (7.29) below forces eval to have the correct domain and codomain.

$$\begin{tikzcd} \mathsf{ob}^{B^A\times A} & \mathsf{ar}^{\mathrm{eval}} \arrow[l, "\mathsf{source}"'] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^{B} \end{tikzcd} \tag{7.29}$$

Expanding this diagram using the required modules is a two stage process, giving

$$\begin{tikzcd}
\mathsf{ob}^{A} & \mathsf{ob}^{B^A}\times\mathsf{ob}^{A} \arrow[l, "p_2"'] \arrow[r, "p_1"] \arrow[d, "\mathsf{prod}"] & \mathsf{ob}^{B^A} & & \\
\mathsf{ar} \arrow[u, "\mathsf{rproj}"] \arrow[dr, "\mathsf{source}"'] & \mathsf{cone} \arrow[l, "\mathsf{lproj}"'] \arrow[r, "\mathsf{rproj}"] & \mathsf{ar} \arrow[u, "\mathsf{lproj}"'] \arrow[dl, "\mathsf{source}"] & \mathsf{ob}^{B}\times\mathsf{ob}^{A} \arrow[ul, "\mathsf{fs}"'] \arrow[r, "p_1"] \arrow[d, "p_2"] & \mathsf{ob}^{A} \\
& \mathsf{ob}^{B^A\times A} & \mathsf{ar}^{\mathrm{eval}} \arrow[l, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^{B} &
\end{tikzcd} \tag{7.30}$$

Diagram (7.31) below forces λf to have the correct domain and codomain.

$$\begin{tikzcd}
& \mathsf{ob}^{C} & \mathsf{ob}^{C^A} \\
\mathsf{ob}^{B\times A} & \mathsf{twovf}^{f} \arrow[l, "\mathsf{tsource}"'] \arrow[u, "\mathsf{ttarget}"] \arrow[r, "\mathsf{lam}"] & \mathsf{curry}^{\lambda f} \arrow[u, "\mathsf{csource}"] \arrow[d, "\mathsf{ctarget}"] \\
& & \mathsf{ob}^{B}
\end{tikzcd} \tag{7.31}$$

Diagram (7.32) below forces Diagram (7.25) to commute.

$$\begin{tikzcd}
\mathsf{ob}^{B\times A} & \mathsf{ar}^{\lambda f\times \mathsf{Id}[A]} \arrow[l, "\mathsf{source}"'] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^{C^A\times A} \\
& \mathsf{ar}_2 \arrow[u, "\mathsf{rfac}"] \arrow[dl, "\mathsf{comp}"'] \arrow[dr, "\mathsf{lfac}"] & \\
\mathsf{ar}^{f} \arrow[uu, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^{C} & \mathsf{ar}^{\mathrm{eval}} \arrow[uu, "\mathsf{source}"'] \arrow[l, "\mathsf{target}"]
\end{tikzcd} \tag{7.32}$$

Diagram (7.33) below ensures that requirement CCC–4 holds.

$$
\begin{tikzcd}
\mathsf{ob}^{C^A \times A} & \mathsf{ar}^{\mathrm{eval}} \arrow[l, "\mathsf{source}"'] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^{C} & & \mathsf{ob}^{B} & \\
\mathsf{ar}^{g \times \mathsf{Id}[A]} \arrow[u, "\mathsf{target}"] \arrow[d, "\mathsf{source}"'] & \mathsf{ar}_2^{\langle \mathrm{eval}, g \times \mathsf{Id}[A] \rangle} \arrow[l, "\mathsf{rfac}"'] \arrow[u, "\mathsf{lfac}"] \arrow[dr, "\mathsf{comp}"] & & & \mathsf{ar}^{g} \arrow[u, "\mathsf{source}"] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^{C^A} \\
\mathsf{ob}^{B \times A} & & \mathsf{ar}^{\mathrm{eval} \circ (g \times \mathsf{Id}[A])} \arrow[ll, "\mathsf{source}"'] \arrow[uu, "\mathsf{target}"] & & \mathsf{curry} \arrow[u, "\mathsf{arrow}"] & \\
& & & & \mathsf{twovf} \arrow[u, "\mathsf{lam}"] \arrow[ull, "\mathsf{arrow}"] \arrow[ullll, "\mathsf{tsource}"] \arrow[uuull, "\mathsf{ttarget}"] &
\end{tikzcd}
\tag{7.33}
$$

7.6.5 Invertibility of λ

It follows from CCC-4 that if $\mathcal{C}$ is any Cartesian closed category corresponding to a model $\mathfrak{C}$, then $\mathfrak{C}(\lambda)$ is a bijection. The Completeness Theorems 8.2.1 and 8.2.4 then imply that there is an arrow λ^{-1} in **CCC** that, as its name suggests, is a formal inverse to λ.

Chapter 8

Graph-based logic

8.1 Assertions in graph-based logic

8.1.1 Definition Let F be an $\mathbf{E}$-form. A diagram of the kind

$$\begin{tikzcd} & D' \arrow[d, "f'"] \\ D'' \arrow[r, "f''"'] & D \end{tikzcd} \tag{8.1}$$

in $\mathsf{SynCat}[\mathbf{E}, F]$ is called a **potential factorization** or **PF**.

8.1.2 Definition Suppose there are morphisms of diagrams

$$\begin{tikzcd} & \delta' \arrow[d, "\phi'"] \\ \delta'' \arrow[r, "\phi''"'] & \delta \end{tikzcd} \tag{8.2}$$

for which

a) δ, δ' and δ'' are all diagrams in $\mathsf{SynCat}[\mathbf{E}, F]$.

b) $D = \mathsf{Lim}[\delta]$, $D' = \mathsf{Lim}[\delta']$ and $D'' = \mathsf{Lim}[\delta'']$.

c) f' is the fill-in arrow induced by ϕ' and f'' is the fill-in arrow induced by ϕ''.

Then Diagram (8.2) is a **description** of the potential factorization (8.1).

By Lemma 4.3.5, the description of a potential factorization can be taken to lie in the graph of the constructor space sketch E that generates $\mathbf{E}$. There are in general many descriptions of a PF. The description is not part of the structure. We do not know of an example of a diagram of the form of (8.1) that we can prove does not have a description.

8.1.3 Notation
We will use suggestive notation for a potential factorization, as exhibited in the diagram below.

$$\begin{tikzcd} & \mathit{claim} \arrow[d, "\mathit{claimcon}"] \\ \mathit{hyp} \arrow[r, "\mathit{hypcon}"'] & \mathit{wksp} \end{tikzcd} \tag{8.3}$$

The names hypcon, $\mathit{claimcon}$ and wksp respectively abbreviate "hypothesis construction", "claim construction" and "workspace". The reason for the names of the arrows and nodes is discussed in 8.5. In many examples, including all those in this monograph, $\mathit{claimcon}$ is monic and corresponds to a formal selection of a subset of those objects formally denoted by wksp.

8.1.4 Actual factorizations

If an actual factorization arrow $\mathit{verif} : \mathit{hyp} \to \mathit{claim}$ can be constructed using the rules of Chapter 4 in $\mathsf{SynCat}[\mathbf{E}, F]$ that makes

$$\begin{tikzcd} & \mathit{claim} \arrow[d, "\mathit{claimcon}"] \\ \mathit{hyp} \arrow[ur, "\mathit{verif}"] \arrow[r, "\mathit{hypcon}"'] & \mathit{wksp} \end{tikzcd} \tag{8.4}$$

commute, then we say that the potential factorization (8.1) is **deducible**. We remind the reader that by 6.5, $\mathsf{SynCat}[\mathbf{E}, F]$ is $\mathsf{FinLimTh}\,[E, \mathsf{Name}[F]]$, so that the rules of Chapter 4 apply.

If for some model $\mathfrak{M}$ of $\mathsf{SynCat}[\mathbf{E}, F]$ there is an arrow ξ of **Set** that makes

$$\begin{tikzcd} & \mathfrak{M}(\mathit{claim}) \arrow[d, "\mathfrak{M}(\mathit{claimcon})"] \\ \mathfrak{M}(\mathit{hyp}) \arrow[ur, "\xi"] \arrow[r, "\mathfrak{M}(\mathit{hypcon})"'] & \mathfrak{M}(\mathit{wksp}) \end{tikzcd} \tag{8.5}$$

commute, then we say the model $\mathfrak{M}$ **satisfies** the potential factorization. If for *every* model $\mathfrak{M}$ there is such an arrow ξ then we say that the potential factorization is **valid**.

We give examples of potential factorizations in Sections 8.3 and 8.4. Section 8.5 discusses the general concept of potential factorization.

8.2 Soundness and completeness

8.2.1 Theorem *In any syntactic category* $\mathsf{SynCat}[\mathbf{E}, F]$*, a potential factorization is deducible if and only if it is valid.*

Proof That deducibility implies validity follows from the fact that functors preserve factorizations.

For the converse, let

$$\begin{tikzcd} & \mathit{claim} \arrow[d, "\mathit{claimcon}"] \\ \mathit{hyp} \arrow[r, "\mathit{hypcon}"'] & \mathit{wksp} \end{tikzcd} \tag{8.6}$$

be a potential factorization in $\mathsf{SynCat}[\mathbf{E}, F]$. Suppose it is valid. Because the functor $\mathrm{Hom}(\mathit{hyp}, -)$ is a model, the hypothesis of the theorem implies that we may choose an arrow ξ of **Set** such that the diagram

$$
\begin{tikzcd}
 & \mathrm{Hom}(\mathit{hyp}, \mathit{claim}) \arrow[d, "\mathfrak{M}(\mathit{hyp}, \mathit{claimcon})"] \\
\mathrm{Hom}(\mathit{hyp}, \mathit{hyp}) \arrow[ur, "\xi"] \arrow[r, "\mathfrak{M}(\mathit{hyp}, \mathit{hypcon})"'] & \mathfrak{M}(\mathit{hyp}, \mathit{wksp})
\end{tikzcd}
\tag{8.7}
$$

commutes. Define $\mathit{verif} := \xi(\mathrm{id}_{\mathit{hyp}})$. Then $\mathit{claimcon} \circ \mathit{verif} = \mathit{hypcon}$, so that the potential factorization is deducible. □

8.2.2 Remark Any model $\mathfrak{M}$ is a functor that preserves finite limits, so that if $\mathit{claimcon}$ is monic, so is $\mathfrak{M}(\mathit{claimcon})$. In that case, $\mathfrak{M}(\mathit{verif})$ necessarily equals ξ.

Section 7.6.5 gives an example of how completeness can be used.

8.2.3 Remarks Proofs of soundness in string-based logic contain an induction which is missing in the preceding argument. In the present system, a theorem can be identified with an arrow of $\mathsf{SynCat}[\mathbf{E}, F]$. Chapter 4 describes the recursive construction of arrows in the finite-limit theory of a sketch and so is the analog of the inductive part of string-based proofs of completeness. We repeat once more that $\mathsf{SynCat}[\mathbf{E}, F]$ is indeed the finite-limit theory of a sketch, namely $\mathsf{FinLimTh}\big[E[\mathsf{Name}[F]]\big]$ (see 6.5) and hence the constructions in Chapter 4 do apply in this case.

8.2.4 Proposition *Let $\delta : I \to \mathsf{SynCat}[\mathbf{E}, F]$ be a diagram. Suppose for every model $\mathfrak{M}$ of $\mathsf{SynCat}[\mathbf{E}, F]$, $\mathfrak{M} \circ \delta$ commutes. Then δ commutes.*

Proof Suppose

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "h"'] & B \arrow[d, "g"] \\
 & C
\end{tikzcd}
\tag{8.8}
$$

is a diagram in $\mathsf{SynCat}[\mathbf{E}, F]$. Then, because $\mathrm{Hom}(A, -)$ is a model,

$$
\begin{tikzcd}
\mathrm{Hom}(A, A) \arrow[r, "\mathrm{Hom}(A, f)"] \arrow[dr, "\mathrm{Hom}(A, h)"'] & \mathrm{Hom}(A, B) \arrow[d, "\mathrm{Hom}(A, g)"] \\
 & \mathrm{Hom}(A, C)
\end{tikzcd}
$$

commutes. By chasing $\mathsf{Id}[A]$ around the diagram both ways, we get $g\circ f = h$, so Diagram (8.8) commutes as well. The general result follows because every diagram can be triangulated.□

8.3 Example: A fact about diagrams in any category

The following proposition holds in any category.

8.3.1 Proposition *In any category, given the following diagram*

$$\begin{tikzcd} A \arrow[r, "h"] \arrow[d, "f"'] & B \arrow[d, "k"] \\ C \arrow[ur, "x"] \arrow[r, "g"'] & D \end{tikzcd} \tag{8.9}$$

if the two triangles commute then so does the outside square.

Proof $k\circ h = k\circ(x\circ f) = (k\circ x)\circ f = g\circ f$.□

Let F be the **Cat**-form (**Cat** = $\mathsf{FinLimTh}[\mathit{Cat}]$) with not necessarily commutative diagrams of the form (8.9) as models. Such a form can be realized by specifying that $\mathsf{Name}[\mathsf{F}]$ be a constant whose type is the limit of Diagram (6.4).

We construct here the potential factorization in $\mathsf{SynCat}[\mathbf{Cat}, \mathsf{F}]$ that corresponds to Proposition 8.3.1. The construction takes place entirely in $\mathsf{FinLimTh}[\mathit{Cat}]$; no reference to the constant $\mathsf{Name}[\mathsf{F}]$ is made, since we are working directly with the description of the type (codomain) of $\mathsf{Name}[\mathsf{F}]$, which is Diagram (6.4).

As we pointed out in Section 6.3, an element of the value in a model of $\mathsf{Lim}[D(6.4)]$ is a diagram such as Diagram (8.9). However, Diagram (6.4) carries only the information as to the source and target of the arrows in Diagram (8.9).

The structure we must actually work with should include the information as to which pairs are composable. There are four composable pairs in Diagram (8.9), and each one inhabits the value of ar_2, the node of formal composable pairs in a category (see Appendix 7.2).

The following diagram thus contains the basic information about sources, targets and composability that are required for stating Proposition 8.3.1, and so its limit is suitable for being the node *wksp* of the proof corresponding to the

Proposition.

$$
\begin{tikzcd}
 & & \mathsf{ar}_2^{\langle g,f\rangle} \arrow[dl, "\mathsf{rfac}"'] \arrow[dr, "\mathsf{lfac}"] & & \\
 & \mathsf{ar}^f \arrow[r, "\mathsf{target}"] \arrow[dl, "\mathsf{source}"'] & \mathsf{ob}^C & \mathsf{ar}^g \arrow[l, "\mathsf{source}"'] \arrow[dr, "\mathsf{target}"] & \\
\mathsf{ob}^A & \mathsf{ar}_2^{\langle x,f\rangle} \arrow[u, "\mathsf{rfac}"] \arrow[r, "\mathsf{lfac}"] & \mathsf{ar}^x \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"] & \mathsf{ar}_2^{\langle k,x\rangle} \arrow[l, "\mathsf{rfac}"] \arrow[d, "\mathsf{lfac}"] & \mathsf{ob}^D \\
 & \mathsf{ar}^h \arrow[ul, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^B & \mathsf{ar}^k \arrow[l, "\mathsf{source}"] \arrow[ur, "\mathsf{target}"'] & \\
 & & \mathsf{ar}_2^{\langle k,h\rangle} \arrow[ul, "\mathsf{rfac}"] \arrow[ur, "\mathsf{lfac}"'] & &
\end{tikzcd}
\tag{8.10}
$$

The statement that the two triangles in Diagram (8.9) commute is: $x \circ f = h$ and $k \circ x = g$. Using the composition arrow $\mathsf{comp} : \mathsf{ar}_2 \to \mathsf{ar}$ of $\mathsf{SynCat}[\mathit{Cat}]$, this statement amounts to saying that Diagram (8.9) is a member of $\mathfrak{C}\big(\mathsf{Lim}[D(8.11)]\big)$

(below) so we take $\mathit{hyp} = \mathsf{Lim}[D(8.11)]$.

$$\begin{tikzcd}
 & & \mathsf{ar}_2^{\langle g,f\rangle} \arrow[dl, "\mathsf{rfac}"'] \arrow[dr, "\mathsf{lfac}"] & & \\
 & \mathsf{ar}^f \arrow[r, "\mathsf{target}"] \arrow[dl, "\mathsf{source}"'] & \mathsf{ob}^C & \mathsf{ar}^g \arrow[l, "\mathsf{source}"'] \arrow[dr, "\mathsf{target}"] & \\
\mathsf{ob}^A & \mathsf{ar}_2^{\langle x,f\rangle} \arrow[u, "\mathsf{rfac}"] \arrow[r, "\mathsf{lfac}"] \arrow[d, "\mathsf{comp}"] & \mathsf{ar}^x \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"] & \mathsf{ar}_2^{\langle k,x\rangle} \arrow[u, "\mathsf{comp}"] \arrow[l, "\mathsf{rfac}"] \arrow[d, "\mathsf{lfac}"] & \mathsf{ob}^D \\
 & \mathsf{ar}^h \arrow[ul, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^B & \mathsf{ar}^k \arrow[l, "\mathsf{source}"] \arrow[ur, "\mathsf{target}"'] & \\
 & & \mathsf{ar}_2^{\langle k,h\rangle} \arrow[ul, "\mathsf{rfac}"] \arrow[ur, "\mathsf{lfac}"'] & &
\end{tikzcd} \tag{8.11}$$

The statement that the outside of Diagram (8.9) commutes is that the diagram

is a member of $\mathfrak{C}\big(\mathsf{Lim}[\delta(8.12)]\big)$, so we take $\mathit{claim} = \mathsf{Lim}[D(8.12)]$:

$$
\begin{tikzcd}
 & & & \mathsf{ar}_2^{\langle g,f\rangle} \arrow[dl, "\mathsf{rfac}"'] \arrow[dr, "\mathsf{lfac}"] \arrow[ddrrr, "\mathsf{comp}"] & & & \\
 & & \mathsf{ar}^f \arrow[r, "\mathsf{target}"'] \arrow[dl, "\mathsf{source}"'] & \mathsf{ob}^C & \mathsf{ar}^g \arrow[l, "\mathsf{source}"] \arrow[dr, "\mathsf{target}"] & & \\
 & \mathsf{ob}^A & \mathsf{ar}_2^{\langle x,f\rangle} \arrow[u, "\mathsf{rfac}"] \arrow[r, "\mathsf{lfac}"] & \mathsf{ar}^x \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"] & \mathsf{ar}_2^{\langle k,x\rangle} \arrow[l, "\mathsf{rfac}"] \arrow[d, "\mathsf{lfac}"] & \mathsf{ob}^D & \mathsf{ar}^{k\circ g\circ f} \\
 & & \mathsf{ar}^h \arrow[ul, "\mathsf{source}"] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^B & \mathsf{ar}^k \arrow[l, "\mathsf{source}"'] \arrow[ur, "\mathsf{target}"] & & \\
 & & & \mathsf{ar}_2^{\langle k,h\rangle} \arrow[ul, "\mathsf{rfac}"] \arrow[ur, "\mathsf{lfac}"] \arrow[uurrr, "\mathsf{comp}"'] & & &
\end{tikzcd}
\tag{8.12}
$$

Diagram (8.10) is a restriction of both Diagram (8.11) and Diagram (8.12). By Lemma 5.3.2, this inclusion induces arrows

$$\mathit{claimcon} : \mathsf{Lim}[D(8.12)] \to \mathsf{Lim}[D(8.10)]$$

and

$$\mathit{hypcon} : \mathsf{Lim}[D(8.11)] \to \mathsf{Lim}[D(8.10)]$$

producing a potential factorization in $\mathsf{FinLimTh}[\mathit{Cat}]$ (hence in $\mathsf{SynCat}[\mathbf{Cat}, F]$, which contains $\mathsf{FinLimTh}[\mathit{Cat}]$ as a subcategory):

$$
\begin{tikzcd}
 & \mathsf{Lim}[D(8.12)] \arrow[d, "\mathit{claimcon}"] \\
\mathsf{Lim}[D(8.11)] \arrow[r, "\mathit{hypcon}"'] & \mathsf{Lim}[D(8.10)]
\end{tikzcd}
\tag{8.13}
$$

This potential factorization expresses the content of Proposition 8.3.1 in diagrammatic form. It should be clear that the node $\mathsf{Lim}[D(8.10)]$ could have been replaced by $\mathsf{Lim}[D(6.4)]$.

In Section 8.7 we construct an arrow *verif* making the diagram

$$\begin{tikzcd} & \mathsf{Lim}[D(8.12)] \arrow[d, "\mathit{claimcon}"] \\ \mathsf{Lim}[D(8.11)] \arrow[ur, "\mathit{verif}"] \arrow[r, "\mathit{hypcon}"'] & \mathsf{Lim}[D(8.10)] \end{tikzcd} \tag{8.14}$$

an actual factorization (see Section 8.1.4).

8.4 Example: A fact about Cartesian closed categories

Proposition 8.3.1 holds in any category. We now discuss a theorem of Cartesian closed categories, to show how the system presented in this monograph handles structure that cannot be expressed using Ehresmann sketches. The latter are equivalent in expressive power to ordinary first order logic. (An excellent presentation of the details of this fact may be found in [Adámek and Rosicky, 1994].) Thus this example in a certain sense requires higher-order logic.

8.4.1 Proposition *In any Cartesian closed category, if*

$$\begin{tikzcd} A \times B \arrow[r, "g"] \arrow[dr, "h"'] & C \arrow[d, "f"] \\ & D \end{tikzcd}$$

commutes, then so does

$$\begin{tikzcd} A \arrow[r, "\lambda g"] \arrow[dr, "\lambda h"'] & C^B \arrow[d, "f^B"] \\ & D^B \end{tikzcd}$$

See Appendix 7.6 for notation. The arrow f^B is defined by

$$f^B := \lambda(C^B \times B \xrightarrow{\text{eval}} C \xrightarrow{f} D) : C^B \longrightarrow D^B \tag{8.15}$$

The proof follows from the fact that λ is invertible and the calculation

$$\begin{aligned}
\mathrm{eval} \circ \big((f^B \circ \lambda g) \times \mathsf{Id}[B]\big) &= \mathrm{eval} \circ \big((f^B \times \mathsf{Id}[B]) \circ (\lambda g \times \mathsf{Id}[B])\big) \\
&= \big(\mathrm{eval} \circ (f^B \times \mathsf{Id}[B])\big) \circ (\lambda g \times \mathsf{Id}[B]) \\
&= (f \circ \mathrm{eval}) \circ (\lambda g \times \mathsf{Id}[B]) \\
&= f \circ (\mathrm{eval} \circ (\lambda g \times \mathsf{Id}[B])) \\
&= f \circ g = h = \mathrm{eval} \circ (\lambda h \times \mathsf{Id}[B])
\end{aligned}$$

The first equality is based on an assertion true in all categories that can be handled in our system in a manner similar to (but more complicated than) that of module (7.12) in Appendix 7.4. The second and fourth equalities are associativity of composition and are proven using Figure (7.2) of Appendix 7.2. The sixth equality is a hypothesis. The other three equalities are all based on Diagram (7.32) in Appendix 7.6.

We present here the potential factorization corresponding to the third equality, which is the most complicated of those based on Diagram (7.32). In this presentation, unlike that of 8.3, we will use the modules developed in Chapter 7 to simplify the figures. The actual factorization corresponding to this potential factorization is given in 8.8.

The fact under discussion is that the diagram

$$\begin{tikzcd}
A \times B \arrow[r, "\lambda g \times \mathsf{Id}[B]"] & C^B \times B \arrow[r, "f^B \times \mathsf{Id}[B]"] \arrow[d, "\mathrm{eval}"'] & D^B \times B \arrow[d, "\mathrm{eval}"] \\
& C \arrow[r, "f"'] & D
\end{tikzcd} \tag{8.16}$$

commutes.

Thus *wksp* will be the limit of the following diagram, which describes the objects and arrows in Diagram (8.16) but has no requirements on its commutativity.

$$\begin{tikzcd}
\mathsf{ar}^{\lambda g \times \mathsf{Id}[B]} \arrow[r, "\mathsf{target}"] \arrow[d, "\mathsf{source}"'] & \mathsf{ob}^{C^B \times B} & \mathsf{ar}^{f^B g \times \mathsf{Id}[B]} \arrow[l, "\mathsf{source}"'] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^{D^B \times B} \\
\mathsf{ob}^{A \times B} & \mathsf{ar}^{eval} \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"'] & & \mathsf{ar}^{eval} \arrow[u, "\mathsf{source}"'] \arrow[d, "\mathsf{target}"] \\
& \mathsf{ob}^{C} & \mathsf{ar}^{f} \arrow[l, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^{D}
\end{tikzcd}$$

(8.17)

We define *hyp* to be the limit of the following diagram, in which $\phi = (f \circ \mathrm{eval}) \circ (\lambda g \times \mathsf{Id}[B])$.

$$
\begin{tikzcd}
 & & \mathsf{ob}^{D^B \times B} & & \\
 & \mathsf{ar}^{f^B \times \mathsf{Id}[B]} \arrow[ur, "\mathsf{target}"] \arrow[dl, "\mathsf{source}"'] & \mathsf{ar}_2^{\langle \mathrm{eval}, f^B \times \mathsf{Id}[B] \rangle} \arrow[l, "\mathsf{rfac}"'] \arrow[rr, "\mathsf{lfac}"] & & \mathsf{ar}^{\mathrm{eval}} \arrow[ull, "\mathsf{source}"'] \arrow[dddl, "\mathsf{target}"] \\
\mathsf{ob}^{C^B \times B} & \mathsf{ar}^{\lambda g \times \mathsf{Id}[B]} \arrow[l, "\mathsf{target}"'] \arrow[r, "\mathsf{source}"] & \mathsf{ob}^{A \times B} & \mathsf{ar}^{\phi} \arrow[l, "\mathsf{source}"'] & \\
\mathsf{ar}^{\mathrm{eval}} \arrow[u, "\mathsf{source}"] \arrow[d, "\mathsf{target}"] & \mathsf{ar}^{f \circ \mathrm{eval}} \arrow[ul, "\mathsf{source}"] & \mathsf{ar}_2^{\langle f \circ \mathrm{eval}, \lambda g \times \mathsf{Id}[B] \rangle} \arrow[l, "\mathsf{lfac}"'] \arrow[ul, "\mathsf{rfac}"'] \arrow[ur, "\mathsf{comp}"'] \arrow[dr, "\mathsf{target}"'] & & \\
\mathsf{ob}^{C} & \mathsf{ar}_2^{\langle f, \mathrm{eval} \rangle} \arrow[ul, "\mathsf{rfac}"] \arrow[u, "\mathsf{comp}"] \arrow[d, "\mathsf{lfac}"] & & \mathsf{ob}^{D} & \\
 & \mathsf{ar}^{f} \arrow[ul, "\mathsf{source}"] \arrow[urr, "\mathsf{target}"'] & & &
\end{tikzcd}
\tag{8.18}
$$

Then Diagram (8.17) is a subdiagram of Diagram (8.18) (the big rectangle in Diagram (8.17) is the perimeter of Diagram (8.18)) and we define *hypcon* to be the induced arrow from $\mathsf{Lim}[D(8.18)]$ to $\mathsf{Lim}[D(8.17)]$.

The object *claim* is the limit of a diagram we shall refer to as Diagram (8.18′), obtained from Diagram (8.18) by adjoining an arrow labeled comp from $\mathsf{ar}_2^{\langle \mathrm{eval}, f^B \times \mathsf{Id}[B] \rangle}$ to $\mathsf{ar}^{f \circ \mathrm{eval}}$. Diagram (8.18′) includes Diagram (8.18) and hence Diagram (8.17), and we take the arrow *claimcon* to be the arrow from Diagram (8.18′) to Diagram (8.17) induced by this inclusion.

We now have a potential factorization

$$\begin{tikzcd}
 & \mathsf{Lim}[D(8.18')] \arrow[d, "\mathit{claimcon}"] \\
\mathsf{Lim}[D(8.18)] \arrow[r, "\mathit{hypcon}"'] & \mathsf{Lim}[D(8.17)]
\end{tikzcd} \tag{8.19}$$

This potential factorization expresses the content of the third equality in the calculation in the proof of Proposition 8.4.1. We provide an actual factorization

$$\begin{tikzcd}
 & \mathsf{Lim}[8.18'] \arrow[d, "\mathit{claimcon}"] \\
\mathsf{Lim}[D(8.18)] \arrow[ur, "\mathit{verif}"] \arrow[r, "\mathit{hypcon}"'] & \mathsf{Lim}[D(8.17)]
\end{tikzcd} \tag{8.20}$$

in Section 8.8.

8.4.2 Remark Diagram (8.20) is a diagram in $\mathbf{CCC}$. Let the $\mathbf{CCC}$-form F be determined by requiring that $\mathsf{Name}[F]$ be a freely adjoined global element with target *wksp*. Then via the embedding $\mathsf{CatTh}[\mathbf{FinLim}, \mathbf{CCC}]$ into $\mathsf{SynCat}[\mathbf{CCC}, F]$, Diagram (8.20) is also a diagram in $\mathsf{SynCat}[\mathbf{CCC}, F]$, and if it has an actual factorization in $\mathsf{CatTh}[\mathbf{FinLim}, \mathbf{CCC}]$ then it also has an actual factorization in $\mathsf{SynCat}[\mathbf{CCC}, F]$.

8.5 Discussion of the examples

8.5.1 General discussion

In a potential factorization

$$\begin{tikzcd}
 & \mathit{claim} \arrow[d, "\mathit{claimcon}"] \\
\mathit{hyp} \arrow[r, "\mathit{hypcon}"'] & \mathit{wksp}
\end{tikzcd} \tag{8.21}$$

each of *hyp*, *claim* and *wksp* represents a type of entity that can be constructed in an $\mathbf{E}$-category, specifically in an arbitrary category for Example 8.3 and in any Cartesian closed category for Example 8.4. In this section, we discuss a way of thinking about these nodes that exhibits how they could represent a theorem about $\mathbf{E}$-categories.

1. The node *wksp* (for "workspace") represents the data involved in the union of the hypothesis and the conclusion. For a given theorem, the choice of

what is actually included in *wksp* may be somewhat arbitrary (see Item 1 in Section 8.5.3 below).

2. The node *hyp* represents possible additional properties that are part of the assumptions in the theorem being represented.

3. The node *claim* represents the properties that the theorem asserts must hold given the assumptions.

4. The arrow *hypcon* represents the selection or construction necessary to see the hypothesis as part of the workspace. In both our examples, *hypcon* represents a simple forgetting of properties.

5. The arrow *claimcon* represents the selection or construction necessary to see the claim as part of the workspace.

6. The arrow *verif* in an actual factorization represents a specific way, *uniform in any model*, that any entity of type *hyp* can be transformed into, or recognized as, an entity of type *claim*.

8.5.2 Discussion of Example 8.3

In Example 8.3,

1. *wksp* represent squares of the form

$$\begin{tikzcd} A \arrow[r, "h"] \arrow[d, "f"'] & B \arrow[d, "k"] \\ C \arrow[ur, "x"] \arrow[r, "g"'] & D \end{tikzcd} \tag{8.22}$$

   with no commutativity conditions.

2. *hyp* represents diagrams of the form of Diagram (8.22) in which the two triangles commute;

3. *claim* represents diagram of the form of Diagram (8.22) in which the outside square commutes.

4. *hypcon* represents forgetting that the two triangles commute. Because it represents forgetting a property in this case, *hypcon* is monic, but in general it need not be.

5. *claimcon* represents forgetting that the outside square commutes.

8.5.3 Discussion of Example 8.4

In Example 8.4,

1. *wksp* represent diagrams of the form of Diagram (8.16) with no commutativity conditions and no recognition that any sequence of arrows is composable. The phrase "The form of Diagram (8.16)" refers to the source and target commonalities of the arrows in the diagram. Obviously, some sequences compose but we have not represented that in *wksp*, although we could have.

2. *hyp* represents diagrams of the form of Diagram (8.16), recognizing the composite $f \circ \mathrm{eval}$ and the fact that $f \circ \mathrm{eval}$, $\lambda g \times \mathsf{Id}[B]$ and eval, $f^B \times \mathsf{Id}[B]$ are composable pairs.

3. *claim* represents diagrams of the form of Diagram (8.16) that commute.

4. *hypcon* represents forgetting the information concerning composition in *hyp*.

5. *claimcon* represents forgetting the information concerning composition in *claim*.

We discuss the meaning of the actual factorization arrows *verif* in 8.9.

8.5.4 Explicit description instead of pattern recognition

The representation of facts such as those of Example 8.3 and 8.4 as potential factorizations is *variable free* in the sense that in each statement, one does not refer to a particular diagram such as Diagram (8.9) or Diagram (8.16) which stands as a pattern for all such diagrams. Propositions 8.3.1 and 8.4.1 state the fact in question using those diagrams as patterns, and understanding their meaning calls on the reader's ability to recognize patterns. Our description of the fact in the examples as a potential factorization is much more complicated because the diagrams involved in the potential factorization are essentially explicit descriptions of the relations between the nodes and arrows of Diagram (8.9) and Diagram (8.16) respectively, relations which a knowledgeable reader grasps from seeing the diagrams without having them indicated explicitly.

Thus at the price of considerably more complexity we have substituted *explicit description* of the structure *implied* by the diagrams in the assertions of Propositions 8.3.1 and 8.4.1. The relation between the explicit description and the written assertions reminds us of the relation between a program in a high-level programming language and the assembly code (or at least lower-level code) of a compiled form of the assertions. We believe this is an important step in the process of implementing on a computer the machinery described in this monograph.

8.6 The rules of graph-based logic

In first-order logic, rules in the form of a context-free grammar are given for constructing terms and formulas, and further rules (rules of inference) are given for deriving formulas from formulas. These rules are intended to preserve truth.

In Section 4.4 we gave rules for constructing all the objects and arrows of $\mathsf{FinLimTh}[S]$ (hence in any syntactic category – see 6.5) for an arbitrary finite-limit sketch S, and for constructing a basis for all the commutative diagrams in $\mathsf{FinLimTh}[S]$. These rules correspond to both the term and formula construction rules and the rules of inference of string-based logic. The tools of a typical string-based logic include constant symbols, variables, function symbols, logical operators and quantifiers. Here we have nodes, arrows and commutative diagrams. (See Remark 8.6.3.) What corresponds to a sentence is a potential factorization as in Diagram (8.21), and what corresponds to the satisfiability of the sentence in a model $\mathfrak{M}$ is the existence of an arrow ξ in that model for which

$$\begin{tikzcd} & \mathfrak{M}(\mathit{claim}) \arrow[d, "\mathfrak{M}(\mathit{claimcon})"] \\ \mathfrak{M}(\mathit{hyp}) \arrow[ur, "\xi"] \arrow[r, "\mathfrak{M}(\mathit{hypcon})"'] & \mathfrak{M}(\mathit{wksp}) \end{tikzcd} \tag{8.23}$$

commutes. A demonstration of the deducibility of the sentence corresponds to the construction of an arrow $\mathit{verif} : \mathit{hyp} \to \mathit{claim}$ in $\mathsf{FinLimTh}[S]$ such that

$$\begin{tikzcd} & \mathit{claim} \arrow[d, "\mathit{claimcon}"] \\ \mathit{hyp} \arrow[ur, "\mathit{verif}"] \arrow[r, "\mathit{hypcon}"'] & \mathit{wksp} \end{tikzcd} \tag{8.24}$$

commutes. Thus the same rules suffice for *constructing* the sentence (the potential factorization) and for *proving* it (constructing the arrow that makes it an actual factorization). See Section 8.5 for further discussion of these points.

8.6.1 Remarks Each rule in Section 4.4 is actually a *rule scheme*, and each instance of the scheme is an assertion that, given certain arrows and commutative diagrams (see Remark 8.6.2 below) in $\mathsf{SynCat}[\mathbf{E}, S]$, other arrows or commutative diagrams exist in $\mathsf{SynCat}[\mathbf{E}, S]$. For example, if δ is a diagram with shape graph

$$\begin{tikzcd} & i \arrow[d, "u"] \\ j \arrow[r, "v"'] & k \end{tikzcd} \tag{8.25}$$

then the following rule is an instance of ∃LIM:

$$
\frac{\begin{tikzcd} & \delta(i) \arrow[d, "\delta(u)"] \\ \delta(j) \arrow[r, "\delta(u)"'] & \delta(k) \end{tikzcd}}{\begin{tikzcd} \mathsf{Lim}[\delta] \arrow[r, "{\mathsf{Proj}[\mathsf{Lim}[\delta], i]}"] \arrow[d, "{\mathsf{Proj}[\mathsf{Lim}[\delta], j]}"'] \arrow[dr, "{\mathsf{Proj}[\mathsf{Lim}[\delta], k]}" description] & \delta(i) \arrow[d] \\ \delta(j) \arrow[r] & \delta(k) \end{tikzcd}} \tag{8.26}
$$

8.6.2 Remark Rules ∃FIA, !FIA and CFIA assume the existence of commutative cones, but a commutative cone is a collection of interrelated commutative diagrams, so the statement above that each scheme assumes the existence of certain arrows and commutative diagrams is correct. Thus given a cone $\Theta : v \lhd (\delta : I \to \mathcal{C})$, an instance of ∃FIA is this rule:

$$
\frac{\begin{tikzcd} \mathsf{Vertex}[\Theta] \arrow[r, "{\mathsf{Proj}[\mathsf{Lim}[\Theta], i]}"] \arrow[d, "{\mathsf{Proj}[\mathsf{Lim}[\Theta], j]}"'] \arrow[dr, "{\mathsf{Proj}[\mathsf{Lim}[\Theta], k]}" description] & \delta(i) \arrow[d] \\ \delta(j) \arrow[r] & \delta(k) \end{tikzcd}}{\begin{tikzcd} \mathsf{Vertex}[\Theta][\Theta'\delta] \arrow[r, "\mathsf{Fillin}"'] & \mathsf{Lim}[\delta] \end{tikzcd}} \tag{8.27}
$$

The point of this remark is that ∃FIA is a rule with a diagram as hypothesis and an arrow as conclusion. The hypothesis is the *cone itself*, not the string "$\Theta : v \lhd (\delta : I \to \mathcal{C})$" or any other description of it.

8.6.3 Remark In the second paragraph of this section we mentioned the tools of string-based logic – strings made from symbols with an implicit structure given by a grammar. The tools of graph-based logic are diagrams made from nodes and arrows. We have been criticized for not giving a rigorous definition of concepts such as nodes, arrows and diagrams. We would like to point out that logic texts generally do not spell out what symbols are (a decidedly subtle question) or what strings are. The difference between the two situations is that string of symbols are more familiar to most logicians than diagrams are. We claim that that is the *only* difference between them.

8.7 Example: Proof of Proposition 8.3.1

We continue Example 8.3 by constructing and thereby deducing the existence of an arrow $\mathit{verif}: \mathsf{Lim}[D(8.11)] \to \mathsf{Lim}[D(8.12)]$ making

$$\begin{tikzcd} & \mathsf{Lim}[D(8.12)] \arrow[d, "\mathit{claimcon}"] \\ \mathsf{Lim}[D(8.11)] \arrow[ur, "\mathit{verif}"] \arrow[r, "\mathit{hypcon}"'] & \mathsf{Lim}[D(8.10)] \end{tikzcd} \tag{8.28}$$

commute.

We first construct Diagram (8.11′) (not shown) by adjoining ar_3 to Diagram (8.11), along with arrows

$$\begin{aligned} &\mathsf{lfac}: \mathsf{ar}_3 \longrightarrow \mathsf{ar}^k \\ &\mathsf{mfac}: \mathsf{ar}_3 \longrightarrow \mathsf{ar}^x \\ &\mathsf{rfac}: \mathsf{ar}_3 \longrightarrow \mathsf{ar}^f \end{aligned}$$

We further construct Diagram (8.11″) by adjoining

$$\begin{aligned} &\langle \mathsf{lfac}, \mathsf{mfac} \rangle : \mathsf{ar}_3 \longrightarrow \mathsf{ar}_2^{\langle k,x \rangle} \\ &\langle \mathsf{mfac}, \mathsf{rfac} \rangle : \mathsf{ar}_3 \longrightarrow \mathsf{ar}_2^{\langle x,f \rangle} \\ &\mathsf{lass}: \mathsf{ar}_3 \longrightarrow \mathsf{ar}^{\langle g,f \rangle} \\ &\mathsf{rass}: \mathsf{ar}_3 \longrightarrow \mathsf{ar}_2^{\langle k,h \rangle} \end{aligned}$$

to Diagram (8.11′). These arrows are defined in Appendix 7.2.

Diagram (8.10) is a base restriction of each of Diagram (8.11′) and Diagram (8.11″), so, we may, using Lemma 5.3.2, choose arrows

$$\phi_1 : \mathsf{Lim}[D(8.11')] \to \mathsf{Lim}[D(8.10)]$$

and

$$\phi_2 : \mathsf{Lim}[D(8.11'')] \to \mathsf{Lim}[D(8.10)]$$

Diagram (8.11) is a dominant subdiagram of Diagram (8.11′) since the latter is obtained from the former by adjoining a limit of a subdiagram together with their projection arrows (lfac, mfac and rfac). Therefore, using Lemma 5.4.5, we may choose an isomorphism ψ_1 making

$$\begin{tikzcd} \mathsf{Lim}[D(38)] \arrow[rr, "\psi_1"] \arrow[dr, "\mathit{hypcon}"'] & & \mathsf{Lim}[38'] \arrow[dl, "\phi_1"] \\ & \mathsf{Lim}[D(37)] & \end{tikzcd} \tag{8.29}$$

commute.

Similarly, Diagram (8.11′) is a dominant subdiagram of Diagram (8.11″) since the latter is obtained from the former by adjoining four fill-in arrows. Therefore by Lemma 5.4.5 we may choose an isomorphism ψ_2 making

$$\begin{tikzcd} \mathsf{Lim}[D(38')] \arrow[rr, "\psi_1"] \arrow[dr, "\phi_1"'] & & \mathsf{Lim}[38''] \arrow[dl, "\phi_2"] \\ & \mathsf{Lim}[D(37)] & \end{tikzcd} \tag{8.30}$$

commute. We then construct Diagram (8.11‴) by adjoining arrows

$$\mathsf{comp} : \mathsf{ar}_2^{\langle g,f\rangle} \to \mathsf{ar}^{k\circ x\circ f}$$

and

$$\mathsf{comp} : \mathsf{ar}_2^{\langle k,h\rangle} \to \mathsf{ar}^{k\circ x\circ f}$$

where $\mathsf{ar}^{k\circ x\circ f}$ is a new node.

Because of associativity (the right diagram in Figure (7.2) of Appendix 7.2), Diagram (8.11‴) extends Diagram (8.11″) by adjoining a commutative cocone, so we may choose an isomorphism $\psi_3 : \mathsf{Lim}[D(8.11'')] \to \mathsf{Lim}[D(8.11''')]$ and an arrow $\phi_3 : \mathsf{Lim}[D(8.11''')] \to \mathsf{Lim}[D(8.10)]$ making

$$\begin{tikzcd} \mathsf{Lim}[D(38'')] \arrow[rr, "\psi_3"] \arrow[dr, "\phi_2"'] & & \mathsf{Lim}[38'''] \arrow[dl, "\phi_3"] \\ & \mathsf{Lim}[D(37)] & \end{tikzcd} \tag{8.31}$$

commute.

Finally, by Lemma 5.3.2, we may choose arrow $\psi_4 : \mathsf{Lim}[D(8.11''')] \to \mathsf{Lim}[D(8.12)]$ making

$$\begin{tikzcd} \mathsf{Lim}[D(38''')] \arrow[rr, "\psi_4"] \arrow[dr, "\phi_3"'] & & \mathsf{Lim}[39] \arrow[dl, "\mathit{claim}"] \\ & \mathsf{Lim}[D(37)] & \end{tikzcd} \tag{8.32}$$

commute.

We next set

$$\mathit{verif} := \psi_4 \circ \psi_3 \circ \psi_2 \circ \psi_1 \tag{8.33}$$

whence the theorem follows.

8.8 Example: Proof of Theorem 8.4.1

In this section, we provide a factorization of the potential factorization described in Section 8.4.

Diagram (8.18′) contains the following as a subdiagram, in which, using Diagram (8.15),

$$\theta = \langle \mathrm{eval}, f^B \times \mathsf{Id}[B] \rangle = \langle \mathrm{eval}, \lambda(f \circ \mathrm{eval}) \times \mathsf{Id}[B] \rangle$$

$$\begin{tikzcd}
\mathsf{ob}^{C^B \times B} & \mathsf{ar}^{f^B \times \mathsf{Id}[B]} \arrow[l, "\mathsf{source}"'] \arrow[r, "\mathsf{target}"] & \mathsf{ob}^{D^B \times B} \\
 & \mathsf{ar}_2^{\theta} \arrow[u, "\mathsf{rfac}"] \arrow[dl, "\mathsf{comp}"'] \arrow[dr, "\mathsf{lfac}"] & \\
\mathsf{ar}^{f \circ \mathrm{eval}} \arrow[uu, "\mathsf{source}"] \arrow[r, "\mathsf{target}"'] & \mathsf{ob}^{D} & \mathsf{ar}^{\mathrm{eval}} \arrow[uu, "\mathsf{source}"'] \arrow[l, "\mathsf{target}"]
\end{tikzcd} \tag{8.34}$$

This diagram is an instance of Diagram (7.32), so it commutes. The arrow comp satisfies Definition 5.4.1, so Lemma 5.4.5 implies that there is an isomorphism $\mathit{verif} : \mathit{hyp} \to \mathit{claim}$. Now the inclusion of Diagram (8.17) into Diagram (8.18) followed by the inclusion of Diagram (8.18) into Diagram (8.18′) is precisely the inclusion of Diagram (8.17) into Diagram (8.18′). It follows that $\mathit{hypcon} \circ \mathit{verif}^{-1} = \mathit{claimcon}$, so that $\mathit{claimcon} \circ \mathit{verif} = \mathit{hypcon}$ as required.

8.9 Discussion of the proofs.

The factorization $\mathit{verif} : \mathsf{Lim}[D(8.11)] \to \mathsf{Lim}[D(8.12)]$ of Diagram (8.13) given in Equation (8.33) constitutes the recognition that if the two triangles commute, then so does the outside square. The fact that verif makes Diagram (8.13) commute is a codification of the fact that if the two triangles commute then so does the outside square *of the same diagram*. In general, the reason we require that actual factorizations be an arrow in the comma category $(\mathsf{SynCat}[\mathbf{E}, F] \downarrow \mathit{wksp})$ instead of merely an arrow from one node to another is to allow us to assert hypotheses and conclusions that share data (in this case the data in Diagram (8.22)).

The factorization $\mathit{verif} : \mathit{hyp} \to \mathit{claim}$ constructed in 8.8 constitutes recognition that $\mathrm{eval} \circ (f^B \times \mathsf{Id}[B]) = f \circ \mathrm{eval}$ via the arrow $\mathsf{comp} : \mathsf{ar}_2^{\langle \mathrm{eval}, f^B \times \mathsf{Id}[B] \rangle} \to \mathsf{ar}^{f \circ \mathrm{eval}}$ in Diagram (8.18′). Because of this, the node $\mathsf{ar}_2^{\langle f \circ \mathrm{eval}, \lambda g \times \mathsf{Id}[B] \rangle}$ could also be labeled $\langle \mathrm{eval} \circ f^B \times \mathsf{Id}[B], \lambda g \times \mathsf{Id}[B] \rangle$. Thus the factorization also exhibits the fact that

$$\mathrm{eval} \circ (f^B \times \mathsf{Id}[B]) \circ (\lambda g \times \mathsf{Id}[B]) = f \circ \mathrm{eval} \circ (\lambda g \times \mathsf{Id}[B])$$

It is clear that there are many alternative formulations of Proposition 8.4.1. For example, instead of first constructing $\mathsf{ar}_2^{f \circ \mathrm{eval}}$ as in Diagram (8.18) (which is *hyp* in this case), we could have constructed a node $\mathsf{ar}_2^{\mathrm{eval} \circ (f^B \times \mathsf{Id}[B])}$ and an arrow

$$\mathsf{comp} : \mathsf{ar}_2^{\langle \mathrm{eval}, (f^B \times \mathsf{Id}[B]) \rangle} \rightarrow \mathsf{ar}_2^{\mathrm{eval} \circ (f^B \times \mathsf{Id}[B])}$$

Then the construction of an arrow

$$\mathsf{comp} : \mathsf{ar}_2^{\langle f, \mathrm{eval} \rangle} \rightarrow \mathsf{ar}^{\mathrm{eval} \circ (f^B \times \mathsf{Id}[B])}$$

would have proved the theorem.

8.10 Discussion of graph-based logic

This formalism, which uses diagrams and mappings between diagrams instead of strings of symbols, perhaps seems unusual from the point of view of symbolic logic. It contrasts with the usual string-based formalism in two ways. On the one hand, our formalism exhibits explicitly much more detail than the string-based approach about the relationships between different parts of the structure. On the other, our formalism is very close to the way it would be represented in a modern object-oriented computer language as compared to string-based formulas. The nodes become objects and the arrows become methods.

Thus the arrow lam of CCC– 3 (in Section 7.6) can be directly represented in a program object as the method that yields the exponential adjoint of an arrow in a Cartesian closed category. In contrast, a formula in first order logic requires a rather sophisticated parser to translate it into a data structure on which a program can operate. Parsing is well-understood, but it results in a computer representation (for example as a tree or as reordered tokens on a stack) that is very different from the formula before it is parsed. What must be represented in the computer is the formation-tree of a formula or term, not the string of symbols that is usually thought of as the formula or term.

In this sense, the appearance that string based logic is simpler than graph-based logic is an illusion: For people to understand the structure of an expression represented as a string requires them to have sophisticated pattern-recognition abilities. For a computer program to operate with such expressions requires an elaborate parser.

Thus the approach via diagrams has some of the advantages (for example, transparent translation into a programming object) and some of the disadvantages (for example, more of the structure is explicit) of assembly language versus high level languages. (See Remarks 2.4.3 a)

Chapter 9

Equational Theories

9.1 Signatures

9.1.1 Expressions and terms

In the description that follows of the terms and equations for a signature, we use a notation that specifies the variables of a term or equation explicitly. In particular, one may specify variables that do not appear in the expression. For this reason, the formalism we introduce in the definitions below distinguishes an *expression* such as $f(x, g(y, x), z)$ from a *term*, which is an expression together with a specified set of typed variables; in this case that set could be for example $\{x, y, z, w\}$. This formalism is equivalent to that of [Goguen and Meseguer, 1982].

9.1.2 Definition A pair (Σ, Ω) of sets together with two functions $\mathsf{Inp} : \Omega \to \mathsf{List}[\Sigma]$ and $\mathsf{Outp} : \Omega \to \Sigma$ is called a **signature**. Given a signature $\mathcal{S} := (\Sigma, \Omega)$, elements of Σ are called the **types** of $\mathcal{S}$ and the elements of Ω are called the **operations** of $\mathcal{S}$.

9.1.3 Notation Given a signature $\mathcal{S} = (\Sigma, \Omega)$, we will denote the set Σ of types by $\mathsf{Types}[\mathcal{S}]$ and the set Ω of operations by $\mathsf{Oprns}[\mathcal{S}]$. For any $f \in \Omega$, the list $\mathsf{Inp}[f]$ is called the **input type list** of f and the type $\mathsf{Outp}[f]$ is the **output type** of f.

9.1.4 Remark The input type list of f is usually called the **arity** of f, and the output type of f is usually called simply the **type** of f.

9.1.5 Definition An operation f of a signature $\mathcal{S}$ is called a **constant** if and only if $\mathsf{Inp}[f]$ is the empty list.

9.1.6 Definition A type γ of a signature $\mathcal{S}$ is said to be **inhabited** if and only if either

a) there is a constant of output type γ in $\mathcal{S}$, or

b) there is an operation f of output type γ for which every type in $\mathsf{Inp}[f]$ is inhabited.

The type γ is said to be **empty** if and only if it is not inhabited.

9.2 Terms and equations

In this section, we define terms and equations in a given signature.

9.2.1 Assumptions

In these definitions, we make the following assumptions, useful for bookkeeping purposes.

A.1 We assume that we are given a signature $\mathcal{S}$ for which $\mathsf{Types}[\mathcal{S}] = \{\sigma^i \mid i \in I\}$ for some ordinal I.
A.2 For each $i \in I$, we assume there is an indexed set $\mathsf{Vbl}[\sigma^i] := \{x^i_j \mid j \in \omega\}$ whose elements are by definition **variables of type** σ^i. In this setting, x^i_j is the jth variable of type σ^i.
A.3 The set of variables is ordered by defining

$$x^i_j < x^k_l :\Leftrightarrow \begin{cases} \text{either} & i < k \\ \text{or} & i = k \text{ and } j < l \end{cases}$$

We also define $\mathsf{Vbl}[\mathcal{S}] := \cup_{i\in\omega} \mathsf{Vbl}[\sigma^i]$.

9.2.2 Definition For any type τ, an **expression of type** τ is defined recursively as follows.

Expr.1 A variable of type τ is an expression of type τ.
Expr.2 If f is an operation with $\mathsf{Inp}[f] = (\gamma^i \mid i \in 1\,..\,n)$ and $\mathsf{Outp}[f] = \tau$, and $(e_i \mid i \in 1\,..\,n)$ is a list of expressions for which each e_i is of type γ^i, then $f(e_i \mid i \in 1\,..\,n)$ is an expression of type τ.

9.2.3 Definition The type of a variable x is denoted by $\mathsf{Type}[x]$, so that in the notation of 9.2.1, $\mathsf{Type}[x^i_j] = \sigma^i$. This notation will be extended to include lists and sets of variables as follows:

1. If $L = \langle x^1_2, x^1_2, x^1_3, x^2_2\rangle$, then $\mathsf{Type}[L] := \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2$.
2. If $W = \{x^1_2, x^1_3, x^2_1, x^3_2\}$, then $\mathsf{Type}[W] := \sigma^1\times\sigma^1\times\sigma^2\times\sigma^3$. (Note that this depends on the ordering given by A.1.)

The type of an expression e will be denoted by $\mathsf{Type}[e]$.

9.2.4 Remark Thus the function Type is overloaded: it may be applied to variables, lists or sets of variables, or expressions, and will in the following be applied to terms and equations as well.

In every case, $\mathsf{Type}[z]$ denotes a single type, never a list or set of types. In contrast, $\mathsf{Types}[z]$, defined in Section 9.1.3, denotes a set of types and $\mathsf{TypeList}[z]$, defined below in Definition 9.2.5, denotes a list of types.

9.2.5 Definition For a given expression e, $\mathsf{VarList}[e]$ is defined recursively by requiring that

VL.1 If x is a variable of type σ, $\mathsf{VarList}[x] := (\sigma)$.
VL.2 If $e = f(e_1, \ldots, e_n)$, then $\mathsf{VarList}[e] := (\mathsf{VarList}[e_1])\cdots(\mathsf{VarList}[e_n])$, the concatenate of the lists $\mathsf{VarList}[e_i]$.

9.2.6 Remark For a given expression e, $\mathsf{VarList}[e]$ is the list of variables in e, in order of appearance in e from left to right, counting repetitions.

9.2.7 Definition $\mathsf{Rng}\big[\mathsf{VarList}[e]\big]$, the set of distinct variables occurring in e, is called the **variable set** of e and denoted by $\mathsf{VarSet}[e]$. The list $(\mathsf{List}[\mathsf{Type}])\big[\mathsf{VarList}[e]\big]$ is called the **type list** of e, denoted by $\mathsf{TypeList}[e]$.

9.2.8 Remark If the kth entry of $\mathsf{VarList}[e]$ is x^i_j, then the kth entry of $\mathsf{TypeList}[e]$ is σ^i.

9.2.9 Example Let e be the expression $f(x, g(y, x), z)$. If x and y are variables of type γ and z is of type τ, then the variable list of e is (x, y, x, z), the variable set is $\{x, y, z\}$ and the type list is $(\gamma, \gamma, \gamma, \tau)$. Using the notation of A.2 and supposing $\gamma = \sigma^1$, $\tau = \sigma^2$, $x = x^1_1$, $y = x^1_2$ and $z = x^2_1$, we have $e = f(x^1_1, g(x^1_2, x^1_1), x^2_1)$ and the following statements hold:

$$\begin{aligned}
\mathsf{VarList}[e] &= (x^1_1, x^1_2, x^1_1, x^2_1)\\
\mathsf{VarSet}[e] &= \{x^1_1, x^1_2, x^2_1\}\\
\mathsf{TypeList}[e] &= (\sigma^1, \sigma^1, \sigma^1, \sigma^2)\\
\mathsf{Type}[e] &= \sigma^1 \times \sigma^1 \times \sigma^1 \times \sigma^2
\end{aligned}$$

9.2.10 Definition A **term** t for a signature $\mathcal{S}$ is determined by the following:

TD.1 A set $\mathsf{Var}[t]$ of typed variables. (It is a set, not a list, but it is ordered by the ordering of A.3 in 9.2.1.)
TD.2 An expression $\mathsf{Expr}[t]$.
TD.3 A type $\mathsf{Type}[t] \in \mathsf{Types}[\mathcal{S}]$.

These data must satisfy the following requirements:

TR.1 $\mathsf{VarSet}\big[\mathsf{Expr}[t]\big] \subseteq \mathsf{Var}[t]$.
TR.2 $\mathsf{Type}[t] = \mathsf{Type}\big[\mathsf{Expr}[t]\big]$.

9.2.11 Notation A given term t will be represented as the list

$$(\mathsf{Expr}[t], \mathsf{Var}[t], \mathsf{Type}[t])$$

9.2.12 Definition Let t be a term. The list $\mathsf{InputTypes}[t]$ is defined to be the list whose ith entry is the type of the ith variable in $\mathsf{Var}[t]$ using the ordering given by A.3 in 9.2.1. Thus if the kth entry of $\mathsf{Var}[t]$ is x^i_j, then the kth entry of $\mathsf{InputTypes}[t]$ is σ^i. Observe that there are no repetitions in $\mathsf{Var}[t]$ but there may well be repetitions in $\mathsf{InputTypes}[t]$.

9.2.13 Remark It follows immediately from Definitions 9.2.3 and 9.2.12 that if $t = (e, V, \tau)$ then

$$\prod \mathsf{InputTypes}[t] = \mathsf{Type}[V]$$

9.2.14 Example Let $e = f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right)$ as in Example 9.2.9, and suppose $\mathsf{Outp}[f] = \sigma^5$. Then there are many terms t with $\mathsf{Expr}[t] = e$, for example

$$t_1 := \left(e, \{x_1^1, x_2^1, x_1^2\}, \sigma^5\right)$$

and

$$t_2 := \left(e, \{x_1^1, x_2^1, x_3^1, x_1^2, x_5^7\}, \sigma^5\right)$$

We have $\mathsf{Type}[t_1] = \mathsf{Type}[t_2] = \sigma^5$ and (for example)

$$\mathsf{InputTypes}[t_1] = (\sigma^1, \sigma^1, \sigma^2)$$

and

$$\mathsf{InputTypes}[t_2] = (\sigma^1, \sigma^1, \sigma^1, \sigma^2, \sigma^7)$$

9.2.15 Definition An **equation** E is determined by a set $\mathsf{Var}[E]$ of typed variables (ordered by our convention) and two expressions $\mathsf{Left}[E]$, $\mathsf{Right}[E]$, for which

ER.1 $\mathsf{Type}\big[\mathsf{Left}[E]\big] = \mathsf{Type}\big[\mathsf{Right}[E]\big]$.
ER.2 $\mathsf{VarSet}\big[\mathsf{Left}[E]\big] \cup \mathsf{VarSet}\big[\mathsf{Right}[E]\big] \subseteq \mathsf{Var}[E]$.

9.2.16 Notation We will write $e =_V e'$ to denote an equation E with $V = \mathsf{Var}[E]$, $e = \mathsf{Left}[E]$ and $e' = \mathsf{Right}[E]$. The notation $\mathsf{Type}[E]$ will denote the common type of $\mathsf{Left}[E]$ and $\mathsf{Right}[E]$.

9.2.17 Example Let e be the expression $f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right)$ of Example 9.2.9. Let $e' := g(x_2^1, x_3^1)$. Then there are many equations with e and e' as left and right sides, for example:

$$E_1 := f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right) =_{\{x_1^1, x_2^1, x_3^1, x_1^2, x_1^3\}} g(x_2^1, x_3^1) \tag{9.1}$$

and

$$E_2 := f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right) =_{\{x_1^1, x_2^1, x_3^1, x_1^2, x_2^5\}} g(x_2^1, x_3^1) \tag{9.2}$$

For later use, we need the following definition:

9.2.18 Definition The **most concrete term** associated with an expression e is defined to be the unique term t with the properties that $\mathsf{Expr}[t] = e$ and $\mathsf{Var}[t] = \mathsf{VarSet}[e]$. The **most concrete equation** associated with two expressions e and e' is defined to be the unique equation E such that $\mathsf{Left}[E] = e$, $\mathsf{Right}[E] = e'$, and $\mathsf{Var}[E] = \mathsf{VarSet}[e] \cup \mathsf{VarSet}[e']$.

9.2.19 Example We continue Example 9.2.17. The most concrete equation associated with the expressions $f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right)$ and $g(x_2^1, x_3^1)$ is

$$f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right) =_{\{x_1^1, x_2^1, x_3^1, x_1^2\}} g(x_2^1, x_3^1) \tag{9.3}$$

The most concrete term associated with $f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right)$ is

$$\left(f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right), \{x_1^1, x_2^1, x_1^2\}, \sigma^5\right)$$

The most concrete term associated with $g(x_2^1, x_3^1)$ is

$$\left(g(x_2^1, x_3^1), \{x_2^1, x_3^1\}, \sigma^5\right)$$

assuming $\mathsf{Outp}[g] = \sigma^5$ (this *must* be true if Equation (9.3) is true.)

9.3 Equational theories

9.3.1 Definition An **equational theory** (S, E) is a signature S together with a set of equations E in S.

This definition provides a concept of a *multisorted* equational theory. Universal algebra originated in the study of single-sorted equational theories.

Our concern is with **multisorted equational logic** (MSEL): a system of valid deduction for formulas in an equational theory.

9.4 Rules of inference of equational deduction

Goguen and Meseguer [1982] prove that the following rules for equational deduction in multisorted equational deduction are sound and complete.

reflexivity $\dfrac{}{e =_V e}.$

symmetry $\dfrac{e =_V e'}{e' =_V e}.$

transitivity $\dfrac{e =_V e' \quad e' =_V e''}{e =_V e''}.$

concretion Given a set V of typed variables, $x \in V$ and an equation $e =_V e'$ such that $x \in V \setminus (\mathsf{VarSet}[e] \cup \mathsf{VarSet}[e'])$, and given that $\mathsf{Type}[x]$ is inhabited,

$$\frac{e =_V e'}{e =_{V \setminus \{x\}} e'}$$

abstraction Given a set V of typed variables and a variable x,

$$\frac{e =_V e'}{e =_{V\cup\{x\}} e'}$$

substitutivity Given a set V of typed variables, $x \in V$, and expressions u and u' for which $\mathsf{Type}[x] = \mathsf{Type}[u] = \mathsf{Type}[u']$,

$$\frac{e =_V e' \quad u =_W u'}{e[x \leftarrow u] =_{V\setminus\{x\}\cup W} e'[x \leftarrow u']}$$

9.5 Deductions in MSEL

We now define a deduction in MSEL of an equation E from a list $(E_1, \ldots, E_n)$ of equations (called **premises** in this context) as a rooted tree. This definition is not as succinct as it could be, but the form we give makes it easy to prove that every deduction corresponds to an actual factorization (Section 12.3).

9.5.1 Definition Let E be an equation and $P := (E_1, \ldots, E_n)$ a list of equations. A **deduction of** E **from** P has one of the following four forms.

D.1 (E), where $P = (E)$.
D.2 (E), where P is the empty list and E is of the form $e =_V e$. (Note that reflexivity is the only rule with empty premises.)
D.3 (E, D), where D is a deduction of an equation E' from P (the same list of premises) and

$$\frac{E'}{E}$$

is an instance of a rule of inference of MSEL.
D.4 (E, D_1, D_2), where for $i = 1, 2$, D_i is a deduction of an equation E_i from a list of premises P_i, $P = P_1P_2$ (the concatenate), and

$$\frac{E_1 \quad E_2}{E}$$

is an instance of a rule of inference of MSEL.

Chapter 10

Signatures to Sketches

We now show how to construct a finite-product sketch S corresponding to a given signature in such a way that the categories of models of the signature and of the sketch are naturally equivalent.

10.1 The sketch associated to a signature

Given a signature $\mathcal{S} = (\Sigma, \Omega)$, we now construct a finite-product sketch $\mathsf{Sk}[\mathcal{S}]$. This sketch, like any finite-product sketch, determines and is determined (up to isomorphism) by a finite-product form F: Precisely (see Chapter 6), there is a diagram $\delta : I \to \mathbf{FinProd}$ and a global element $\mathsf{Name}[\mathsf{F}] : 1 \to \mathsf{Lim}[\delta]$ in $\mathsf{SynCat}[\mathbf{FinProd}, \mathsf{F}]$ with the property that the value of $\mathsf{Name}[\mathsf{F}]$ in the initial model of $\mathsf{SynCat}[\mathbf{FinProd}, \mathsf{F}]$ in **Set** consists (up to isomorphism) of the graph, diagrams and (discrete) cones that make up the sketch S. Moreover, the finite-product theory $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ (defined in [Barr and Wells, 1999], Section 7.5) is equivalent as a category to the finite-product category $\mathsf{CatTh}\big[\mathbf{FinProd}, \mathsf{F}\big]$.

10.2 The graphs and cones of $\mathsf{Sk}[\mathcal{S}]$

In what follows, we recursively define arrows and commutative diagrams in $\mathsf{Sk}[\mathcal{S}]$ associated to terms and equations of $\mathcal{S}$ respectively.

10.2.1 Definition The set of nodes of $\mathsf{Sk}[\mathcal{S}]$ consist by definition of the following:

OS.1 Each type of $\mathcal{S}$ is a node.
OS.2 Each list $v = (\gamma_i \mid i \in 1\,.\,.\,n)$ that is the input type list (see Remark 9.1.3) of at least one operation in Ω is a node.

10.2.2 Definition The arrows of $\mathsf{Sk}[\mathcal{S}]$ consist by definition of the following:

AS.1 Each operation f in Ω is an arrow $f : \mathsf{Inp}[f] \to \mathsf{Outp}[f]$.
AS.2 For each list $v = (\gamma_i \mid i \in 1\,.\,.\,n)$ that is the input type list of some operation in Ω, there is an arrow $\mathsf{Proj}[i] : v \to \gamma_i$ for each $i \in 1\,.\,.\,n$. (We will write $\mathsf{Proj}[v, i]$ for $\mathsf{Proj}[i]$ if necessary to to avoid confusion, and on the other hand we will write p_i for $\mathsf{Proj}[i]$ in some diagrams to save space.)

10.2.3 Definition The cones of $\mathsf{Sk}[\mathcal{S}]$ consist by definition of the following: For each list $v = (\gamma_1, \ldots, \gamma_n)$ that is the input type list of some operation in Ω, there is a cone of $\mathsf{Sk}[\mathcal{S}]$ with vertex v and an arrow $\mathsf{Proj}[i] : v \to \gamma_i$ for each $i \in 1\,.\,.\,n$.

It follows that in a model M of the sketch $\mathsf{Sk}[\mathcal{S}]$, $M(v) = \prod_{i \in 1..n} M(\gamma_i)$.

10.2.4 Constants

If the signature contains constants, then one of the lists mentioned in OS.2 is the empty list. As a consequence, the sketch will contain an empty cone by Definition 10.2.3, and the vertex will become a terminator in a model.

10.2.5 Lemma *If σ is an inhabited type of $\mathcal{S}$, then there is a constant of type σ (global element of σ) in $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$.*

Proof The proof is an easy structural induction.□

10.3 Terms as arrows

We now describe how to associate each term of a signature $\mathcal{S}$ to an arrow in $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ and each equation to a commutative diagram or a pair of equal arrows in $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$. The constructions given here are an elaboration of those in [Barr and Wells, 1999], pages 185–186.

10.3.1 The arrow in $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ corresponding to a term

We first define recursively two arrows $\mathsf{Sep}[e]$ and $\mathsf{Par}[e]$ of $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ for each expression e, and an arrow $\mathsf{Dia}[t]$ of $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ for each term t. The arrow

$$\mathsf{Arr}[t] := \mathsf{Sep}\big[\mathsf{Expr}[t]\big] \circ \mathsf{Par}\big[\mathsf{Expr}[t]\big] \circ \mathsf{Dia}[t] : \prod \mathsf{InputTypes}[t] \to \mathsf{Outp}[\mathsf{Expr}[t]]$$

will then be the arrow of $\mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ associated with t; the meaning of the term t in a model of the signature is up to equivalence the same function as the value of $\mathsf{Arr}[t]$ in the corresponding model of $\mathsf{Sk}[\mathcal{S}]$.

In these definitions, we suppress mention of the universal model of $\mathsf{Sk}[\mathcal{S}]$. For example, if the universal model is $\mathsf{UnivMod}[\mathcal{S}] : \mathsf{Sk}[\mathcal{S}] \to \mathsf{FPTh}\big[\mathsf{Sk}[\mathcal{S}]\big]$ and Θ is a node of $\mathsf{Sk}[\mathcal{S}]$, then we write Θ instead of $\mathsf{UnivMod}[\Theta]$. We treat arrows of $\mathsf{Sk}[\mathcal{S}]$ similarly.

10.3.2 Definition For an expression e, $\mathsf{Sep}[e]$ is defined recursively by these requirements:

Sep.1 If e is a variable x of type τ, then $\mathsf{Sep}[e] := \mathsf{Id}[\tau]$. Using the notation introduced in Section 9.2.1, A.2, if $e = x^i_j$, then $\mathsf{Sep}[e] := \mathsf{Id}[\sigma^i]$.

Sep.2 Suppose $e = f(e_i \mid i \in 1\,..\,n)$, where f is an operation with $\mathsf{Inp}[f] = (\gamma_i \mid i \in 1\,..\,n)$ and $\mathsf{Outp}[f] = \tau$. By definition, for $i \in (1\,..\,n)$, $\mathsf{Type}[e_i] = \gamma_i$. Then $\mathsf{Sep}[e]$ is defined to be the arrow

$$\prod_{i=1}^n \mathsf{Dom}[\mathsf{Sep}\big[[e_i]\big] \xrightarrow{\displaystyle\prod_{i=1}^{n} \mathsf{Sep}[e_i]} \prod_{i=1}^n \gamma_i \xrightarrow{\;f\;} \tau \tag{10.1}$$

10.3.3 Remarks (a) Sep.2 recursively constructs the correct parenthesization of the domain of $\mathsf{Sep}[e]$, as illustrated in Examples 10.3.9 and Section 11.4.
(b) If $n = 0$ in Sep.2, in other words $\mathsf{Inp}[f]$ is empty, the composite in Diagram (10.2) becomes

$$1 \longrightarrow 1 \xrightarrow{f} \tau$$

(c) "Sep" is short for "separated", so named because $\mathsf{Sep}[e]$ represents the expressions e with variables renamed so that no duplicates occur. It might have been better to refer to this as the "linearization" of e, but we were afraid this would cause confusion with linear sketches.

10.3.4 Definition For an expression e, $\mathsf{Par}[e]$ is defined recursively by these requirements:

Par.1 If e is a constant of type τ, then $\mathsf{Par}[e] := \mathsf{Id}[1]$, the formal identity of the formal terminal object.
Par.2 If e is a variable x of type τ, then $\mathsf{Par}[e] := \mathsf{Id}[\tau]$.
Par.3 If $e = f(e_i \mid i \in 1\,..\,n)$ as in Sep.2, then

$$\prod \mathsf{TypeList}[e] \xrightarrow{\mathsf{Ass}[e]} \prod_{i=1}^{n} \left(\prod \mathsf{TypeList}[e_i]\right) \xrightarrow{\prod\limits_{i=1}^{n} \mathsf{Par}[e_i]} \prod_{i=1}^{n} \mathrm{cod}[\mathsf{Par}[e_i]] \tag{10.2}$$

where $\mathsf{Ass}[e]$ is the canonical associativity arrow.

10.3.5 Remarks (a) $\mathsf{Par}[e]$ is so called because it parenthesizes $\prod_{i=1}^{n} \mathsf{TypeList}[e]$.
(b) Note that the definition ensures that $\mathsf{Par}[e]$ is an identity arrow or a product of canonical associativity arrows.

10.3.6 Definition Let t be an arbitrary term. Then

$$\mathsf{Dia}[t] : \prod \mathsf{InputTypes}[t] \to \prod \mathsf{TypeList}\big[\mathsf{Expr}[t]\,\big]$$

is defined to be the unique arrow induced by requiring that the following diagrams commute for each pair

$$(i, k) \in \big(1\,..\,\mathsf{Length}[\mathsf{InputTypes}[t]]\big) \times \big(1\,..\,\mathsf{Length}[\mathsf{TypeList}\,[\mathsf{Expr}[t]\,]]\big)$$

with the property that the ith variable from the left in $\mathsf{Expr}[t]$ is $(\mathsf{Var}[t])_k$.

$$\begin{tikzcd}
\prod \mathsf{InputTypes}[t] \arrow[rr, "\mathsf{Dia}[t]"] \arrow[dr, "\mathsf{Proj}[k]"'] & & \prod \mathsf{TypeList}\big[\mathsf{Expr}[t]\,\big] \arrow[dl, "\mathsf{Proj}[i]"] \\
& (\mathsf{InputTypes}[t])_k &
\end{tikzcd} \tag{10.3}$$

Alternatively, suppose $\mathsf{Var}[t]$ has cardinality L and $\mathsf{VarList}\big[\mathsf{Expr}[t]\big]$ has length M. Let $\phi : 1\,..\,M \to 1\,..\,L$ be defined by $\phi(m) = l$ if $(\mathsf{Var}[t])_l = (\mathsf{VarList}\big[\mathsf{Expr}[t]\big])_m$ (there is a unique l that makes this true). Then we may also define

$$\mathsf{Dia}[t] : \prod \mathsf{InputTypes}[t] \to \prod \mathsf{TypeList}\big[\mathsf{Expr}[t]\big]$$

to be the arrow $(\mathsf{Proj}[\phi(m)] \mid m \in 1\,..\,M)$.

This works because the $(\phi(m))$th type in $\prod \mathsf{TypeList}\big[\mathsf{Expr}[t]\big]$ is indeed the type of the $(\phi(m))$th variable in $\mathsf{VarSet}\big[\mathsf{Expr}[t]\big]_m$ (see Definition 9.2.5).

This is equivalent to requiring the diagrams (10.3) to commute. The two definitions are useful for different sorts of calculations and are therefore included.

10.3.7 Remark $\mathsf{Dia}[t]$ is in some sense a generalized diagonal map; hence the name.

10.3.8 Example Consider $e := g(x_1^1, c)$, where c is a constant of type σ^2 and g has type σ^3. Suppose

$$t = (g(x_1^1, c), \{x_1^1, x_1^4\}, \sigma^3)$$

Then t corresponds to the arrow

$$\sigma^1 \times \sigma^4 \underbrace{\xrightarrow{\mathsf{Proj}[1]}}_{\mathsf{Dia}[t]} \sigma^1 \underbrace{\xrightarrow{\langle \mathsf{Id}[\sigma^1], !\rangle}}_{\mathsf{Par}[e]} \underbrace{\sigma^1 \times 1 \xrightarrow{\mathsf{Id}[\sigma^1]\times c} \sigma^1 \times \sigma^2 \xrightarrow{g} \sigma^3}_{\mathsf{Sep}[e]}$$

Note that one does not have to consider constants in constructing $\mathsf{Dia}[t]$.

10.3.9 Example Let $e := f\left(x_1^1, g(x_2^1, x_1^1), x_1^2\right)$ with $\mathsf{Inp}[g] = (\sigma^1, \sigma^1)$, $\mathsf{Outp}[g] = \sigma^2$. $\mathsf{Inp}[f] = (\sigma^1, \sigma^2, \sigma^2)$, and $\mathsf{Outp}[f] = \sigma^5$. Let

$$t := (e, \{x_1^1, x_2^1, x_1^2, x_3^4\}, \sigma^5)$$

Then

$$\mathsf{VarList}\big[\mathsf{Expr}[t]\big] = (x_1^1, x_2^1, x_1^1, x_1^2)$$
$$\mathsf{InputTypes}[t] = (\sigma^1, \sigma^1, \sigma^2, \sigma^4)$$
$$\mathsf{Var}[t] = \{x_1^1, x_2^1, x_1^2, x_3^4\}$$
$$\mathsf{TypeList}\big[\mathsf{Expr}[t]\big] = (\sigma^1, \sigma^1, \sigma^1, \sigma^2)$$

and

$$\mathsf{Type}[\mathsf{Var}[t]] = \prod \mathsf{InputTypes}[t] = \sigma^1 \times \sigma^1 \times \sigma^2 \times \sigma^4$$

If we use the first definition of $\mathsf{Dia}[t]$ in Definition 10.3.6, then the following four triangles must commute:

$$\begin{tikzcd}
\sigma^1\times\sigma^1\times\sigma^2\times\sigma^4 \arrow[rr, "\mathsf{Dia}[t]"] \arrow[dr, "\mathsf{Proj}[1]"'] & & \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2 \arrow[dl, "\mathsf{Proj}[1]"] & \sigma^1\times\sigma^1\times\sigma^2\times\sigma^4 \arrow[rr, "\mathsf{Dia}[t]"] \arrow[dr, "\mathsf{Proj}[2]"'] & & \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2 \arrow[dl, "\mathsf{Proj}[2]"] \\
& \sigma^1 & & & \sigma^1 & \\
\sigma^1\times\sigma^1\times\sigma^2\times\sigma^4 \arrow[rr, "\mathsf{Dia}[t]"] \arrow[dr, "\mathsf{Proj}[1]"'] & & \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2 \arrow[dl, "\mathsf{Proj}[3]"] & \sigma^1\times\sigma^1\times\sigma^2\times\sigma^4 \arrow[rr, "\mathsf{Dia}[t]"] \arrow[dr, "\mathsf{Proj}[3]"'] & & \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2 \arrow[dl, "\mathsf{Proj}[4]"] \\
& \sigma^1 & & & \sigma^2 &
\end{tikzcd} \tag{10.4}$$

It follows that $\mathsf{Dia}[t]$ is given by the following diagram, where to save space we write p_k for $\mathsf{Proj}[k]$.

$$\sigma^1\times\sigma^1\times\sigma^2\times\sigma^4 \xrightarrow{\langle p_1, p_2, p_1, p_3\rangle} \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2 \tag{10.5}$$

and that $\mathsf{Arr}[t]$ is the composite

$$\begin{array}{rcl}
 & \sigma^1\times\sigma^1\times\sigma^2\times\sigma^4 & \\
\langle p_1,p_2,p_1,p_3\rangle & \Big\downarrow & \Big\}\,\mathsf{Dia}[t] \\
 & \sigma^1\times\sigma^1\times\sigma^1\times\sigma^2 & \\
\langle p_1,\langle p_2,p_3\rangle,p_4\rangle & \Big\downarrow & \Big\}\,\mathsf{Par}[e] \\
 & \sigma^1\times(\sigma^1\times\sigma^1)\times\sigma^2 & \\
\mathsf{Id}[\sigma^1]\times g\times\mathsf{Id}[\sigma^2] & \Big\downarrow & \\
 & \sigma^1\times\sigma^2\times\sigma^2 & \Bigg\}\,\mathsf{Sep}[e] \\
f & \Big\downarrow & \\
 & \sigma^5 &
\end{array}$$

10.3.10 Remark The first stage of recursion in constructing the third node in the preceding diagram gives

$$\mathsf{Dom}\big[\mathsf{Sep}[\mathsf{Id}[\sigma^1]]\big]\times\mathsf{Dom}\big[\mathsf{Sep}[g(x_2^1,x_1^1)]\big]\times\mathsf{Dom}\big[\mathsf{Sep}[\mathsf{Id}[\sigma^1]]\big]$$

10.3.11 Remark The definition of

$$\mathsf{Arr}[t]:\prod\mathsf{InputTypes}[t]\to\mathsf{Outp}[\mathsf{Expr}[t]]$$

determines a unique arrow, once a choice of the product $\prod\mathsf{InputTypes}[t]$ is made. Changing the choice of the products occurring in the intermediate stages of the definition, namely $\prod_{i=1}^n\mathsf{Dom}[\mathsf{Sep}\big[[e_i]\big]$ (in the case where recursion is required) and $\prod_{i=1}^n\mathsf{TypeList}[e]$ do not change $\mathsf{Arr}[t]$ because $\mathsf{Par}[e]$ is the unique associativity isomorphism, which commutes with the isomorphisms between different choices of products.

10.4 The finite-product sketch associated with an equational theory

Let the equation $E := e =_V e'$ be given. Define the terms t_1 and t_2 by $t_1 = (e,\mathsf{Var}[E],\mathsf{Type}[E])$ and $t_2 = (e',\mathsf{Var}[E],\mathsf{Type}[E])$ (using the notation of 9.2.10). Recall that $\mathsf{Type}[E] = \mathsf{Type}[e] = \mathsf{Type}[e']$. The notation $\mathsf{InputTypes}[E]$ will denote the list $\mathsf{InputTypes}[t_1]$, which is the same as $\mathsf{InputTypes}[t_2]$. As in 10.3.1,

we have arrows $\mathsf{Arr}[t_1]$ and $\mathsf{Arr}[t_2]$ with the same domain and codomain. We will associate the diagram

$$\mathsf{InputTypes}[E] \underset{\mathsf{Arr}[t_2]}{\overset{\mathsf{Arr}[t_1]}{\rightrightarrows}} \mathsf{Type}[E] \tag{10.6}$$

to the equation E. By 10.3.1, this is the same as

$$\begin{tikzcd}
\mathsf{InputTypes}[E] \arrow[r, "\mathsf{Dia}[t_1]"] \arrow[d, "\mathsf{Dia}[t_2]"'] & \mathsf{TypeList}[e] \arrow[d, "\mathsf{Sep}[t_1] \circ \mathsf{Par}[t_1]"] \\
\mathsf{TypeList}[e'] \arrow[r, "\mathsf{Sep}[t_2] \circ \mathsf{Par}[t_2]"'] & \mathsf{Type}[E]
\end{tikzcd} \tag{10.7}$$

This completes the translation.

10.4.1 Remark The commutative diagram as exhibited above can also be viewed as a pair of formally equal arrows as in Diagram (10.6), and in what follows we will use this description frequently.

10.4.2 Definition The finite-product sketch associated with an equational theory consists by definition of the sketch associated with the signature of the theory (defined in Section 10.1), to which is adjoined the diagram just defined that is associated to each equation of the theory.

[**To do:** ***At this point we need to show how to construct the equational theory associated with each finite-product sketch. See the note at the end of Section 1.3.***]

Chapter 11

Substitution

As terms are defined recursively, substitution may be defined either by structural recursion or using composition. These two ways of defining substitution are convenient for different purposes. Here we establish the equivalence of the two procedures. It may be useful to the reader to compare the following constructions to the examples in Section 11.4.

11.1 Recursive definition

We define substitution in expressions, then in terms.

Given an expression e, the result of substituting an expression u for the variable x in e is denoted by $e[x \leftarrow u]$ and is defined in this way:

S.1 If c is a constant, $c[x \leftarrow u] := c$.
S.2 If x is a variable, $x[x \leftarrow u] := u$.
S.3 If x and y are different variables, $x[y \leftarrow u] := x$.
S.4 If $e = f(e_1, \ldots, e_n)$, then

$$e[x \leftarrow u] := f\left(e_1[x \leftarrow u], \ldots, e_n[x \leftarrow u]\right)$$

Now let $t := (e, V, \sigma)$ be a term. The expression $t\,[x \leftarrow (u, W, \tau)]$ denotes the result of substituting the term (u, W, τ) for x in t. This expression is defined only if $\mathsf{Type}[x] = \tau$, and it is defined in this way:

$$t\,[x \leftarrow (u, W, \tau)] := \big(e\,[x \leftarrow (u, W, \tau)]\,, (V \setminus \{x\}) \cup W, \tau\big)$$

In particular,

$$\mathsf{Arr}\big[t\,[x \leftarrow (u, W, \tau)]\big] := \mathsf{Arr}\big[\big(e\,[x \leftarrow (u, W, \tau)]\,, (V \setminus \{x\}) \cup W, \tau\big)\big]$$

11.2 Direct definition

The alternative way suggested by Examples 11.4.1 and 11.4.2 is to define $\mathsf{Arr}\big[t\,[x \leftarrow (u, W, \tau)]\big]$ directly, given

$$\mathsf{Arr}\,[e, V, \sigma] = \mathsf{Sep}[e]\,\mathsf{Par}[e]\,\mathsf{Dia}[(e, V, \sigma)] \tag{11.1}$$

$$\mathsf{Arr}\,[u, W, \tau] = \mathsf{Sep}[u]\,\mathsf{Par}[u]\,\mathsf{Dia}[(u, W, \tau)] \tag{11.2}$$

Note that $\mathsf{Arr}\,[e, V \cup W, \sigma] = \mathsf{Sep}[e]\,\mathsf{Par}[e]\,\mathsf{Dia}[(e, V \cup W, \sigma)]$, so that $\mathsf{Arr}\,[e, V \cup W, \sigma]$ differs from $\mathsf{Arr}\,[e, V, \sigma]$ only in the D-composand of the arrow.

We have

$$\mathsf{Dom}\big[D\,[(e, V \cup W, \sigma)]\big] = \mathsf{Type}\,[V \cup W] = \prod \mathsf{InputTypes}\,[e, V \cup W, \sigma]$$

Now we define an arrow

$$\mathsf{Insert}\,[u, x, V, W] : \mathsf{Type}\big[(V \setminus \{x\}) \cup W\big] \to \mathsf{Type}\,[V \cup W]$$

It is defined whenever V and W are sets of variables, x is a variable, and u is an expression with $\mathsf{Type}[u] = \mathsf{Type}[x]$. If $x \notin V$ we take $\mathsf{Insert}\,[u, x, V, W]$ to be the identity arrow. Otherwise, choose $I \in 1\,..\,\mathsf{Length}[V \cup W]$ such that $(V \cup W)_I = x$. Then, for all $i \in \big(1\,..\,\mathsf{Length}\,(V \cup W)\big) \setminus \{I\}$,

$$\mathsf{Insert}\,[u, x, V, W]_i := \begin{cases} \mathsf{Arr}\,[u, (V \setminus \{x\}) \cup W, \tau] & \text{if } i = I \\ \mathsf{Proj}[i-1] & \text{if } i > I \\ \mathsf{Proj}[i] & \text{otherwise} \end{cases}$$

Note that $\mathsf{Type}\big[(V \setminus \{x\}) \cup W\big]$ and $\mathsf{Type}\,[V \cup W]$ can differ in at most one factor depending on whether $x \in W$ or not.

Finally, we define

$$\mathsf{Arr}\big[t\,[x \leftarrow (u, W, \tau)]\big] := \mathsf{Arr}\,[e, V \cup W, \sigma] \circ \mathsf{Insert}\,[u, x, V, W]$$

We have given two methods of obtaining the arrow corresponding to the term for which substitution has been made. It remains to be seen that these two methods give the same arrow.

11.3 Proof of the equivalence of the two constructions

The proof will be by structural induction.

S.1 If f is an arrow that factors through a terminal object, then $f \circ g$ also factors through the terminal object for any g, in particular for Insert.

S.2 $t = (x, V, \sigma)$ and $\sigma = \tau$. We note that

$$\begin{aligned} \mathsf{Arr}[x, V \cup W, \sigma] &= \mathsf{Sep}[x] \circ \mathsf{Par}[x] \circ \mathsf{Dia}[x, V \cup W, \sigma] \\ &= \mathsf{Id}[\sigma] \circ \mathsf{Id}[\sigma] \circ \mathsf{proj}[I] \quad \text{(where } (V \cup W)_I = x\text{)} \\ &= \mathsf{proj}[I] \end{aligned}$$

From the direct definition we have

$$\begin{aligned} \mathsf{Arr}\big[t[x \leftarrow (u, W, \tau)]\big] &= \mathsf{Arr}\,[x, V \cup W, \tau] \circ \mathsf{Insert}\,[u, x, V, W] \\ &= \mathsf{proj}[I] \circ \mathsf{Insert}\,[u, x, V, W] \\ &= \mathsf{Arr}\big[u, (V \setminus \{x\}) \cup W, \tau\big] \end{aligned}$$

by the definition of $\mathsf{Insert}\,[u, x, V, W]$, which agrees with the recursive definition.

S.3 If $x \notin V$, $\mathsf{Insert}[u, x, V, W]$ is the identity arrow by definition. If $x \in V$, then as in the proof for S.2 above, $\mathsf{Arr}[x, V \cup W, \sigma] = \mathsf{Proj}[I]$, and since $y \neq x$, y has a different index in $V \cup W$, so that $\mathsf{Proj}[I] \circ \mathsf{Insert}[u, y, V, W] = \mathsf{Proj}[I]$.

S.4 In this case, we assume $t = (f(e_1, \ldots, e_n), V, \sigma)$, where $\mathsf{Outp}[f] = \sigma$ and for all $i \in 1\,..\,n$, $\mathsf{Outp}[e_i] = \gamma_i$.

To begin with, suppose we have a term t' that is just like t except for the set of variables, so that

$$t' := (f(e_1, \ldots, e_n), V', \sigma)$$

where $\mathsf{VarSet}\,[f(e_1, \ldots, e_n)] \subseteq V'$. For each $i = 1, \ldots, n$, we require that the following diagram commute:

$$\begin{tikzcd}
\prod \mathsf{InputTypes}[t'] \arrow[d, "{\mathsf{Dia}[t']}"'] \arrow[ddr, "\phi"] & \\
\prod \mathsf{TypeList}[t'] \arrow[d, "{\mathsf{Ass}[e]}"'] & \\
\prod_{i=1}^n \mathsf{TypeList}[e_i] \arrow[r, "{\mathsf{Proj}[i]}"'] \arrow[dd, "\alpha"'] & \mathsf{TypeList}[e_i] \arrow[d, "{\mathsf{Par}[e_i]}"] \\
 & \mathsf{Dom}[\mathsf{Sep}[e_i] \arrow[d, "{\mathsf{Sep}[e_i]}"] \\
\prod_{i=1}^n \gamma_i \arrow[r, "{\mathsf{Proj}[i]}"'] \arrow[d, "f"'] & \gamma_i \\
\tau &
\end{tikzcd} \tag{11.3}$$

Some observations about these diagrams:

a) $\prod \mathsf{TypeList}[t'] = \prod \mathsf{TypeList}[t] = \prod \mathsf{TypeList}[f(e_1, \ldots, e_n)]$.

b) $\alpha = \prod_{i=1}^n \mathsf{Sep}[e_i] \circ \mathsf{Par}[e_i] = \big\langle \mathsf{Sep}[e_1] \circ \mathsf{Par}[e_1] \circ \mathsf{Proj}[e_1], \ldots, \mathsf{Sep}[e_n] \circ \mathsf{Par}[e_n] \circ \mathsf{Proj}[e_n] \big\rangle$.

c) $\phi = \mathsf{Dia}[e_i, V', \sigma]$.

It follows that

$$\begin{aligned}
\mathsf{Arr}[t'] &= f \circ \big\langle \mathsf{Sep}[e_1] \circ \mathsf{Par}[e_1] \circ \mathsf{Proj}[1], \ldots, \mathsf{Sep}[e_n] \circ \mathsf{Par}[e_n] \circ \mathsf{Proj}[e_n] \big\rangle \circ \mathsf{Ass}[e] \circ \mathsf{Dia}[t'] \\
&= f \circ \big\langle \mathsf{Sep}[e_1] \circ \mathsf{Par}[e_1] \circ \mathsf{Dia}[e_1, V', \gamma_1], \ldots, \mathsf{Sep}[e_n] \circ \mathsf{Par}[e_n] \circ \mathsf{Dia}[e_n, V', \gamma_n] \big\rangle \\
&= f \circ \big\langle \mathsf{Arr}[e_1, V', \gamma_1], \ldots, \mathsf{Arr}[e_n, V', \gamma_n] \big\rangle \qquad (11.4)
\end{aligned}$$

Now we return to our assumption that $t = (f(e_1, \ldots, e_n), V, \sigma)$. By induction hypothesis, we have, for all $i \in 1 \,.\,.\, n$,

$$\mathsf{Arr}\big[e_i[x \leftarrow u], (V \setminus \{x\}) \cup W, \gamma_i\big] = \mathsf{Arr}\big[e_i, V \cup W, \gamma_i\big] \circ \mathsf{Insert}\,[u, x, V, W]$$

By the direct definition, $\mathsf{Arr}\big[t\,[x \leftarrow (u, W, \tau)]\big]$ is

$$\begin{aligned}
&\mathsf{Arr}\,[f(e_1 \ldots, e_n), V \cup W, \sigma] \circ \mathsf{Insert}\,[u, x, V, W] \\
&\quad = f \circ \big\langle \mathsf{Arr}\,[e_1, V \cup W, \gamma_1]\,, \ldots, \mathsf{Arr}\,[e_n, V \cup W, \gamma_n] \big\rangle \circ \mathsf{Insert}\,[u, x, V, W] \\
&\qquad\qquad \text{(by (11.4))} \\
&\quad = f \circ \big\langle \mathsf{Arr}\,[e_1, V \cup W, \gamma_1] \circ \mathsf{Insert}\,[u, x, V, W]\,, \\
&\qquad\qquad \ldots, \mathsf{Arr}\,[e_n, V \cup W, \gamma_n] \circ \mathsf{Insert}\,[u, x, V, W] \big\rangle \\
&\quad = f \circ \big\langle \mathsf{Arr}\,[e_1[e \leftarrow u], (V \setminus \{x\}) \cup W, \gamma_1]\,, \ldots, \mathsf{Arr}\,[e_n[e \leftarrow u], (V \setminus \{x\}) \cup W, \gamma_n] \big\rangle \\
&\qquad\qquad \text{(by induction hypothesis)} \\
&\quad = \mathsf{Arr}\big[f\,(e_1[x_1 \leftarrow u], \ldots, e_n[x_n \leftarrow u])\,, (V \setminus \{x\}) \cup W, \sigma\big] \\
&\qquad\qquad \text{(by (11.4))}
\end{aligned}$$

which is $\mathsf{Arr}\big[t\,[x \leftarrow (u, W, \tau)]\big]$ by the recursive definition. This completes the proof of the equivalence of the two definitions.

Later, we shall use the equivalence of these two methods of obtaining the arrow corresponding to the term in which substitution has been made. To facilitate reference we record this in the form of a lemma.

11.3.1 Lemma *Let* $t := (e, V, \sigma)$ *and* $t' := (u, W, \tau)$ *be terms, suppose* $x \in V$ *and suppose* $\mathsf{Outp}[u] = \mathsf{Type}[x]$ *so that* u *may be substituted for* x*. Then*

$$\mathsf{Arr}\big[t\,[x \leftarrow u]\,, (V \setminus \{x\}) \cup W, \tau\big] = \mathsf{Arr}\big[(e, V \cup W, \sigma)\big] \circ \mathsf{Insert}[u, x, V, W]$$

11.4 Two extended examples of the constructions

11.4.1 Example

$$e := f\left(x_1^1, x_3^4, x_2^3, x_1^1, g(x_1^1, x_2^3), x_1^2\right)$$

$$\begin{cases} \mathsf{Inp}[f] = \sigma^1 \times \sigma^4 \times \sigma^3 \times \sigma^1 \times \sigma^5 \times \sigma^2 \\ \mathsf{Outp}[f] = \sigma^5 \end{cases}$$

$$\begin{cases} \mathsf{Inp}[g] = \sigma^1 \times \sigma^3 \\ \mathsf{Outp}[g] = \sigma^5 \end{cases}$$

$$V = \{x_1^1, x_2^1, \underline{x}_3^1, x_1^2, \underline{x}_2^2, \underline{x}_1^3, x_2^3, x_3^4\}$$

The underlined variables are redundant; that is, they do not appear in the expression e.

$$\begin{cases} u := h(x_1^2, x_2^3) \\ \mathsf{Inp}[h] = \sigma^2 \times \sigma^3 \\ \mathsf{Outp}[h] = \sigma^3 \end{cases}$$

$$W := \{\underline{x}_1^1, x_1^2, \underline{x}_2^2, \underline{x}_3^2, \underline{x}_1^3, x_2^3, \underline{x}_3^3\}$$

x_2^3 is a variable for which we are making a substitution. We wish to calculate

$$(e, V, \sigma^5)\left[x_2^3 \leftarrow (u, W, \sigma^3)\right] = \left(e[x_2^3 \leftarrow u], (V \setminus \{x_2^3\}) \cup W, \sigma^5\right)$$

By direct calculation,

$$(V \setminus \{x_2^3\}) \cup W = \{x_1^1, x_2^1, \underline{x}_3^1, x_1^2, \underline{x}_2^2, \underline{x}_3^2, \underline{x}_1^3, x_2^3, \underline{x}_3^3, x_3^4\}$$

and

$$e(x_2^3 \leftarrow u) = f\left(x_1^1, x_3^4, e(x_1^2, x_2^3), x_1^1, g(x_2^1, e(x_1^2, x_2^3)), x_1^2\right)$$

We now exhibit the arrows for e and u over V: $e = f\left(x_1^1, x_3^4, x_2^3, x_1^1, g(x_1^1, x_2^3), x_1^2\right)$:

$$
\begin{array}{rcl}
& \sigma^1\times\sigma^1\times\underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\sigma^3\times\sigma^4 & \\
\langle p_1, p_8, p_7, p_1, p_2, p_7, p_4\rangle & \Big\downarrow & \Big\}\ \mathsf{Dia}[e] \\
& \sigma^1\times\sigma^4\times\sigma^3\times\sigma^1\times\sigma^1\times\sigma^3\times\sigma^2 & \\
\langle p_1, p_2, p_3, p_4, \langle p_5, p_6\rangle, p_7\rangle & \Big\downarrow & \Big\}\ \mathsf{Par}[e] \\
& \sigma^1\times\sigma^4\times\sigma^3\times\sigma^1\times(\sigma^1\times\sigma^3)\times\sigma^2 & \\
\mathsf{Id}[\sigma^1]\times\mathsf{Id}[\sigma^4]\times\mathsf{Id}[\sigma^3]\times\mathsf{Id}[\sigma^1]\times g\times\mathsf{Id}[\sigma^2] & \left.\begin{array}{c} \Big\downarrow \\ \sigma^1\times\sigma^4\times\sigma^3\times\sigma^1\times\sigma^5\times\sigma^2 \\ f\ \Big\downarrow \end{array}\right\} & \mathsf{Sep}[e] \\
& \sigma^5 &
\end{array}
$$

$u = h(x_1^2, x_2^3)$ (over W):

$$
\begin{array}{rcl}
& \underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\sigma^3\times\underline{\sigma}^3 & \\
\langle p_2, p_6\rangle & \Big\downarrow & \Big\}\ \mathsf{Dia}[u] \\
& \sigma^2\times\sigma^3 & \\
\langle p_1, p_2\rangle = \mathsf{Id}[\sigma^2\times\sigma^3] = \mathsf{Id}[\sigma^2]\times\mathsf{Id}[\sigma^3] & \Big\downarrow & \Big\}\ \mathsf{Par}[u] \\
& \sigma^2\times\sigma^3 & \\
h & \Big\downarrow & \Big\}\ \mathsf{Sep}[u] \\
& \sigma^3 &
\end{array}
$$

Therefore $\mathsf{Arr}[u, W, \sigma^3] = h\langle p_1, p_2\rangle\langle p_2, p_6\rangle = h\langle p_2, p_6\rangle$ and $u = h(x_1^2, x_2^3)$

(over $\big(V \setminus \{x_2^3\}\big) \cup W$) is the arrow

$$\begin{array}{cl}
\underline{\sigma}^1\times\underline{\sigma}^1\times\underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\sigma^3\times\underline{\sigma}^3\times\underline{\sigma}^4 & \\
{\scriptstyle\langle p_4, p_8\rangle}\Big\downarrow & \Big\}\ \mathsf{Dia}[u] \\
\sigma^2\times\sigma^3 & \\
{\scriptstyle\langle p_1, p_2\rangle = \mathsf{Id}[\sigma^2\times\sigma^3] = \mathsf{Id}[\sigma^2]\times\,\mathsf{Id}[\sigma^3]}\Big\downarrow & \Big\}\ \mathsf{Par}[u] \\
\sigma^2\times\sigma^3 & \\
{\scriptstyle h}\Big\downarrow & \Big\}\ \mathsf{Sep}[u] \\
\sigma^3 &
\end{array}$$

so that $\mathsf{Arr}[u, \big(V \setminus \{x_2^3\}\big) \cup W, \sigma^3] = h\langle p_1, p_2\rangle\langle p_4, p_8\rangle = h\langle p_4, p_8\rangle$.

Now suppose that σ^1, σ^2, σ^3 and σ^4 are inhabited. Then by Lemma 10.2.5, we may choose constants $c_i : 1 \to \sigma^i$ for $i \in 1\,..\,4$. Then we have the maps

$$\alpha : \mathsf{Type}[W] \to \mathsf{Type}\big[\big(V \setminus \{x_2^3\}\big) \cup W\big]$$
$$\beta : \mathsf{Type}\big[\big(V \setminus \{x_2^3\}\big) \cup W\big] \to \mathsf{Type}[W]$$

This is α:

$$\begin{array}{c}
\mathsf{Type}[W] = \underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\sigma^3\times\underline{\sigma}^3 \\
\Big\downarrow\ \alpha = \langle c_1!, c_1!, c_1!, p_2, c_2!, c_3!, c_3!, p_6, c_3!, c_4!\rangle \\
\underline{\sigma}^1\times\underline{\sigma}^1\times\underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\sigma^3\times\underline{\sigma}^3\times\underline{\sigma}^4
\end{array}$$

where the codomain is

$$\mathsf{Type}\big[\big(V \setminus \{x_2^3\}\big) \cup W\big]$$

and ! is the unique map with domain $\mathsf{Type}[W]$ and codomain 1. The map β is similarly defined. Both are the maps given by Lemma 12.1.1.

It follows that

$$\langle p_4, p_8\rangle\alpha = \langle p_2, p_6\rangle$$

and

$$\langle p_2, p_6\rangle\beta = \langle p_4, p_8\rangle$$

and that

$$\mathsf{Arr}[u, W, \sigma^3] = \mathsf{Arr}\big[u, \big(V \setminus \{x_2^3\}\big) \cup W, \sigma^3\big]\circ\alpha$$

and

$$\mathsf{Arr}\big[u, \big(V \setminus \{x_2^3\}\big) \cup W, \sigma^3\big] = \mathsf{Arr}[u, W, \sigma^3]\circ\beta$$

We now proceed with our example. After substitution,

$$e[x_2^3 \leftarrow u] = f\left(x_1^1, x_3^4, h(x_1^2, x_2^3), x_1^1, g(x_2^1, h(x_1^2, x_2^3)), x_1^2\right)$$

This corresponds to the arrow

$$
\begin{array}{rcl}
 & \sigma^1\times\sigma^1\times\underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\sigma^3\times\underline{\sigma}^3\times\sigma^4 & \\
\langle p_1, p_{10}, p_4, p_8, p_1, p_2, p_4, p_8, p_4\rangle & \Bigg\downarrow & \Bigg\}\ \mathsf{Dia}[e] \\
 & \sigma^1\times\sigma^4\times\sigma^2\times\sigma^3\times\sigma^1\times\sigma^1\times\sigma^2\times\sigma^3\times\sigma^2 & \\
\langle p_1, p_2, \langle p_3, p_4\rangle, p_5, \langle p_6, \langle p_7, p_8\rangle\rangle, p_9\rangle & \Bigg\downarrow & \Bigg\}\ \mathsf{Par}[e] \\
 & \sigma^1\times\sigma^4\times(\sigma^2\times\sigma^3)\times\sigma^1\times\left(\sigma^1\times(\sigma^2\times\sigma^3)\right)\times\sigma^2 & \\
\mathsf{Id}[\sigma^1]\times\mathsf{Id}[\sigma^4]\times\mathsf{Id}[\sigma^2\times\sigma^3]\times\mathsf{Id}[\sigma^1]\times\left(\mathsf{Id}[\sigma^1]\times h\right)\times\mathsf{Id}[\sigma^2] & \Bigg\downarrow & \\
 & \sigma^1\times\sigma^4\times(\sigma^2\times\sigma^3)\times\sigma^1\times(\sigma^1\times\sigma^3)\times\sigma^2 & \Bigg\}\ \mathsf{Sep}[e] \\
\mathsf{Id}[\sigma^1]\times\mathsf{Id}[\sigma^4]\times h\times\mathsf{Id}[\sigma^1]\times g\times\mathsf{Id}[\sigma^2] & \Bigg\downarrow & \\
 & \sigma^1\times\sigma^4\times\sigma^3\times\sigma^1\times\sigma^5\times\sigma^2 & \\
f & \Bigg\downarrow & \\
 & \sigma^5 &
\end{array}
$$

We now calculate

$$\begin{aligned}
e\left[x_2^3 \leftarrow u\right] &= f\left(\mathsf{Id}[\sigma^1]\times \mathsf{Id}[\sigma^4]\times h\times \mathsf{Id}[\sigma^1]\times g\times \mathsf{Id}[\sigma^3]\right)\\
&\quad\circ\left(\mathsf{Id}[\sigma^1]\times \mathsf{Id}[\sigma^4]\times \mathsf{Id}[\sigma^2\times\sigma^3]\times \mathsf{Id}[\sigma^1]\times(\mathsf{Id}[\sigma^1]\times h)\times \mathsf{Id}[\sigma^2]\right)\\
&\quad\circ\langle p_1, p_2, \langle p_3, p_4\rangle, p_5, \langle p_6, \langle p_7, p_8\rangle\rangle, p_9\rangle\\
&\quad\circ\langle p_1, p_{10}, p_4, p_8, p_1, p_2, p_4, p_8, p_4\rangle\\
&= f\left(\mathsf{Id}[\sigma^1]\times \mathsf{Id}[\sigma^4]\times \mathsf{Id}[\sigma^3]\times \mathsf{Id}[\sigma^1]\times g\times \mathsf{Id}[\sigma^2]\right)\\
&\quad\circ\langle p_1, p_2, h\langle p_3, p_4\rangle, p_8, \langle p_6, h\langle p_7, p_8\rangle\rangle, p_9\rangle\\
&\quad\circ\langle p_1, p_{10}, p_4, p_8, p_1, p_2, p_4, p_8, p_4\rangle\\
&= f\left(\mathsf{Id}[\sigma^1]\times \mathsf{Id}[\sigma^4]\times \mathsf{Id}[\sigma^3]\times \mathsf{Id}[\sigma^1]\times g\times \mathsf{Id}[\sigma^2]\right)\\
&\quad\circ\langle p_1, p_2, p_3, p_4, \langle p_5, p_6\rangle, p_7\rangle\\
&\quad\circ\langle p_1, p_2, h\langle p_3, p_4\rangle, p_5, p_6, h\langle p_7, p_8\rangle, p_9\rangle\\
&\quad\circ\langle p_1, p_{10}, p_4, p_8, p_1, p_2, p_4, p_8, p_4\rangle\\
&= f\left(\mathsf{Id}[\sigma^1]\times \mathsf{Id}[\sigma^4]\times \mathsf{Id}[\sigma^3]\times \mathsf{Id}[\sigma^1]\times g\times \mathsf{Id}[\sigma^2]\right)\\
&\quad\circ\langle p_1, p_2, p_3, p_4, \langle p_5, p_6\rangle, p_7\rangle\\
&\quad\circ\langle p_1, p_{10}, h\langle p_4, p_8\rangle, p_1, p_2, h\langle p_4, p_8\rangle, p_4\rangle\\
&= f\left(\mathsf{Id}[\sigma^1]\times \mathsf{Id}[\sigma^4]\times \mathsf{Id}[\sigma^3]\times \mathsf{Id}[\sigma^1]\times g\times \mathsf{Id}[\sigma^2]\right)\\
&\quad\circ\langle p_1, p_2, p_3, p_4, \langle p_5, p_6\rangle, p_7\rangle\\
&\quad\circ\langle p_1, p_{10}, p_8, p_1, p_2, p_8, p_4\rangle\\
&\quad\circ\langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, h\langle p_4, p_8\rangle, p_9, p_{10}\rangle\\
&= \mathsf{Arr}\left[e, V\cup W, \mathsf{Type}[e]\right]\\
&\quad\circ\left\langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, \mathsf{Arr}\left[u, \left(V\setminus\{x_2^3\}\right)\cup W, \mathsf{Type}[u]\right], p_9, p_{10}\right\rangle
\end{aligned}$$

11.4.2 Example

$$u = m(x_1^2, x_1^2, x_4^4)$$

$$\begin{cases}\mathsf{Inp}[u] = \sigma^2\times\sigma^2\times\sigma^4\\ \mathsf{Outp}[u] = \sigma^3\end{cases}$$

$$W = \{\underline{x}_1^1, x_1^2, \underline{x}_2^2, \underline{x}_3^2, \underline{x}_1^3, \underline{x}_3^3, x_4^4\}$$

This is different from Example 11.4.1 because the variable x_2^3 (in e) for which we are making the substitution does not reappear in u.

This is the arrow for $u = m(x_1^2, x_1^2, x_4^4)$ over W:

$$
\begin{tikzcd}
\sigma^1 \times \sigma^2 \times \underline{\sigma}^2 \times \underline{\sigma}^2 \times \underline{\sigma}^3 \times \underline{\sigma}^3 \times \sigma^4 \arrow[d, "\langle p_1, p_1, p_7 \rangle"'] & {} \arrow[d, phantom, "\Big\}\ \mathsf{Dia}[u]"] \\
\sigma^2 \times \sigma^2 \times \sigma^4 \arrow[d, "\langle p_1, p_2, p_3 \rangle = \mathsf{Id}[\sigma^2 \times \sigma^2 \times \sigma^4] = \mathsf{Id}[\sigma^1] \times \mathsf{Id}[\sigma^2] \times \mathsf{Id}[\sigma^4]"'] & {} \arrow[d, phantom, "\Big\}\ \mathsf{Par}[u]"] \\
\sigma^2 \times \sigma^2 \times \sigma^4 \arrow[d, "m"'] & {} \arrow[d, phantom, "\Big\}\ \mathsf{Sep}[u]"] \\
\sigma^5 & {}
\end{tikzcd}
$$

We calculate

$$\big(V \setminus \{x_2^3\}\big) \cup W = \{x_1^1, x_2^1, \underline{x}_3^1, x_1^2, \underline{x}_2^2, \underline{x}_3^2, \underline{x}_1^3, \underline{x}_3^3, x_3^4, x_4^4\}$$

Then $u = m(x_1^2, x_1^2, x_4^4)$ (over $\big(V \setminus \{x_2^3\}\big) \cup W$) gives the arrow

$$
\begin{tikzcd}
\sigma^1 \times \sigma^1 \times \underline{\sigma}^1 \times \sigma^2 \times \underline{\sigma}^2 \times \underline{\sigma}^2 \times \underline{\sigma}^3 \times \underline{\sigma}^3 \times \sigma^4 \times \sigma^4 \arrow[d, "\langle p_4, p_4, p_{10} \rangle"'] & {} \arrow[d, phantom, "\Big\}\ \mathsf{Dia}[u]"] \\
\sigma^2 \times \sigma^2 \times \sigma^4 \arrow[d, "\langle p_1, p_2, p_3 \rangle = \mathsf{Id}[\sigma^2 \times \sigma^2 \times \sigma^4] = \mathsf{Id}[\sigma^1] \times \mathsf{Id}[\sigma^2] \times \mathsf{Id}[\sigma^4]"'] & {} \arrow[d, phantom, "\Big\}\ \mathsf{Par}[u]"] \\
\sigma^2 \times \sigma^2 \times \sigma^4 \arrow[d, "m"'] & {} \arrow[d, phantom, "\Big\}\ \mathsf{Sep}[u]"] \\
\sigma^5 & {}
\end{tikzcd}
$$

After substitution,

$$e[x_2^3 \leftarrow u] = f\big(x_1^1, x_3^4, m\left(x_1^2, x_1^2, x_4^4\right), x_1^1, g\left(x_2^1, m\left(x_1^2, x_1^2, x_4^4\right)\right), x_1^2\big)$$

This corresponds to the arrow shown below.

$$
\begin{array}{c}
\sigma^1\times\sigma^1\times\underline{\sigma}^1\times\sigma^2\times\underline{\sigma}^2\times\underline{\sigma}^2\times\underline{\sigma}^3\times\underline{\sigma}^3\times\sigma^4\times\sigma^4\\
\left.\Big\downarrow\ \langle p_1,p_9,p_{10},p_4,p_4,p_{10},p_1,p_2,p_4,p_4,p_{10},p_2\rangle\ \right\}\mathsf{Dia}[e]\\
\sigma^1\times\sigma^4\times\sigma^2\times\sigma^2\times\sigma^4\times\sigma^1\times\sigma^1\times\sigma^2\times\sigma^2\times\sigma^4\times\sigma^2\\
\left.\Big\downarrow\ \Big\langle p_1,p_2,\langle p_3,p_4,p_5\rangle\,,p_6,\langle p_7,\langle p_8,p_9,p_{10}\rangle\rangle\,,p_{11}\Big\rangle\ \right\}\mathsf{Par}[e]\\
\sigma^1\times\sigma^4\times\left(\sigma^2\times\sigma^2\times\sigma^2\right)\times\sigma^1\times\left(\sigma^1\left(\sigma^2\times\sigma^2\times\sigma^4\right)\right)\times\sigma^2\\
\Big\downarrow\ \mathsf{Id}[\sigma^1]\times\mathsf{Id}[\sigma^4]\times\mathsf{Id}\left[\sigma^2\times\sigma^2\times\sigma^4\right]\times\mathsf{Id}[\sigma^2]\left(\mathsf{Id}[\sigma^1]\times h\right)\times\mathsf{Id}[\sigma^2]\\
\left.\begin{array}{c}
\sigma^1\times\sigma^4\times\left(\sigma^2\times\sigma^2\times\sigma^2\right)\times\sigma^1\times\left(\sigma^1\times\underline{\sigma}^3\right)\times\sigma^2\\
\Big\downarrow\ \mathsf{Id}[\sigma^1]\times\mathsf{Id}[\sigma^4]\times h\times\mathsf{Id}[\sigma^1]\times g\times\mathsf{Id}[\sigma^2]\\
\sigma^1\times\sigma^4\times\underline{\sigma}^3\times\sigma^1\times\underline{\sigma}^5\times\sigma^2
\end{array}\right\}\mathsf{Sep}[e]\\
\Big\downarrow f\\
\tau
\end{array}
$$

We next re-express this in a convenient form:

$$\begin{aligned}
e[x_2^3 \leftarrow u] &= f \circ \big(\mathsf{Id}[\sigma^1] \times \mathsf{Id}[\sigma^4] \times h \times \mathsf{Id}[\sigma^2] \times g \times \mathsf{Id}[\sigma^2]\big) \\
&\quad \circ \big(\sigma^1 \times \sigma^4 \times \mathsf{Id}\left[\sigma^2 \times \sigma^2 \times \sigma^4\right] \times \mathsf{Id}[\sigma^1] \times \big(\mathsf{Id}[\sigma^1] \times h\big) \times \mathsf{Id}[\sigma^2]\big) \\
&\quad \circ \big\langle p_1, p_2, \langle p_3, p_4, p_5\rangle, p_6, \langle p_7, \langle p_8, p_9, p_{10}\rangle\rangle, p_{11}\big\rangle \\
&\quad \circ \big\langle p_1, p_9, p_4, p_4, p_{10}, p_1, p_2, p_4, p_4, p_{10}, p_2\big\rangle \\
&= f \circ \big(\mathsf{Id}[\sigma^1] \times \mathsf{Id}[\sigma^4] \times \mathsf{Id}[\sigma^3] \times \mathsf{Id}[\sigma^1] \times g \times \mathsf{Id}[\sigma^2]\big) \\
&\quad \circ \big\langle p_1, p_2, h\langle p_3, p_4, p_5\rangle, p_6, \langle p_7, h\langle p_8, p_9, p_{10}\rangle\rangle, p_{11}\big\rangle \\
&\quad \circ \big\langle p_1, p_9, p_4, p_4, p_{10}, p_1, p_2, p_4, p_4, p_{10}, p_2\big\rangle \\
&= f \circ \big(\mathsf{Id}[\sigma^1] \times \mathsf{Id}[\sigma^4] \times \mathsf{Id}[\sigma^3] \times \mathsf{Id}[\sigma^1] \times g \times \mathsf{Id}[\sigma^2]\big) \\
&\quad \circ \big\langle p_1, p_9, h\langle p_4, p_4, p_{10}\rangle, p_1, \langle p_2, h\langle p_4, p_4, p_{10}\rangle\rangle, p_2\big\rangle \\
&= f \circ \big(\mathsf{Id}[\sigma^1] \times \mathsf{Id}[\sigma^4] \times \mathsf{Id}[\sigma^3] \times \mathsf{Id}[\sigma^1] \times g \times \mathsf{Id}[\sigma^2]\big) \\
&\quad \circ \big\langle p_1, p_2, p_3, p_4, \langle p_5, p_6\rangle, p_7\big\rangle \\
&\quad \circ \big\langle p_1, p_{10}, p_8, p_1, p_2, p_8, p_4\big\rangle \\
&\quad \circ \big\langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, h\langle p_4, p_4, p_{10}\rangle, p_9, p_1, p_{11}\big\rangle \\
&= \mathsf{Arr}\left(e, V \cup W, \sigma^5\right) \\
&\quad \circ \big\langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, \mathsf{Arr}\big(u, \big(V \setminus \{x_2^3\}\big) \cup W\big), p_9, p_{10}, p_{11}\big\rangle
\end{aligned}$$

11.4.3 Remark In Examples 11.4.1 and 11.4.2 we may define a map

$$A(e,u) : \prod \mathsf{TypeList}\big[\big(V \setminus \{x_2^3\}\big) \cup W\big] \to \prod \mathsf{TypeList}\,[V \cup W]$$

as follows. Choose $I \in 1\,..\,\mathsf{Length}\,[V \cup W]$ such that

$$\big(\mathsf{TypeList}[V \cup W]\big)_I = x_2^3$$

We next define for all $i \in 1\,..\,\mathsf{Length}\,[V \cup W]$

$$(A(e,u))_i = \mathsf{Proj}[i]$$

and

$$(A(e,u))_I = \mathsf{Arr}\big(u, \big(V \setminus \{x_2^3\}\big) \cup W, \sigma^5\big)$$

With this definition in the previous two examples we have

$$\mathsf{Arr}\big[\big(e, V, \sigma^5\big)\left[x_2^3 \leftarrow \big(u, W, \sigma^3\big)\right]\big] = \mathsf{Arr}\left[e, V \cup W, \sigma^5\right] \circ A(e,u)$$

This is an example of the construction in 11.2.

Chapter 12

Logic of equational theories

12.1 A lemma

The following lemma will be used later in our discussion of the rules "concretion" and "abstraction" that have to do with including extraneous variables in and excluding them from the list of variables of some term. The proof may best be understood by considering the examples in Section 11.

12.1.1 Lemma *Let e be an expression of type τ, and let V_1 and V_2 be lists of variables such that* $\mathsf{VarSet}[e] \subseteq V_1$ *and* $\mathsf{VarSet}[e] \subseteq V_2$*. Let*

$$t_1 := [e, V_1, \tau]$$

and

$$t_2 := [e, V_2, \tau]$$

Then there are arrows

$$\alpha_{12} : \prod \mathsf{TypeList}[t_1] \to \prod \mathsf{TypeList}[t_2]$$

and

$$\alpha_{21} : \prod \mathsf{TypeList}[t_2] \to \prod \mathsf{TypeList}[t_1]$$

for which $\mathsf{Arr}[t_1] = \mathsf{Arr}[t_2] \circ \alpha_{12}$ *and* $\mathsf{Arr}[t_2] = \mathsf{Arr}[t_1] \circ \alpha_{21}$*.*

Proof Assume $\mathsf{Type}[x] = \sigma$. The arrow α_{12} is defined by the following requirements:

α.1 If $x \in V_2 \backslash \mathsf{VarSet}[e]$ and x is in the jth place in $\mathsf{Var}[t_2]$ (so that $\mathsf{TypeList}[t_2]_j = \sigma$), then the diagram

$$\begin{tikzcd}
\prod \mathsf{TypeList}[t_1] \arrow[r, "!"] \arrow[d, "\alpha_{12}"'] & 1 \arrow[d, "c_j"] \\
\prod \mathsf{TypeList}[t_2] \arrow[r, "\mathsf{Proj}[j]"'] & \sigma
\end{tikzcd} \tag{12.1}$$

must commute, where c_j is some constant of type σ (there must be one by Lemma 10.2.5). Note that it does not matter which constant of type σ is chosen.

$\alpha.2$ If $x \in \mathsf{VarSet}[e]$ and x is in the ith place in $\mathsf{Var}[t_1]$ and in the jth place in $\mathsf{Var}[t_2]$, then the diagram

$$\begin{tikzcd}
\prod \mathsf{TypeList}[t_1] \arrow[rr, "\alpha_{12}"] \arrow[dr, "\mathsf{Proj}[i]"'] & & \prod \mathsf{TypeList}[t_2] \arrow[dl, "\mathsf{Proj}[j]"] \\
& \sigma &
\end{tikzcd} \tag{12.2}$$

must commute.

□

12.2 Rules of inference of MSEL as factorizations

In this section we show how the rules of inference of multisorted equational logic can be codified into our present system. This is a two-step process. First, we show that for each rule of inference the pair of equal arrows corresponding to the conclusion of the rule of inference can be constructed using the rules of construction of graph-based logic from the single arrow or the product of the equal pairs of arrows that form the hypothesis of that rule of inference. Next, we exhibit the construction as an actual factorization, where the nodes and arrows appearing in the positions corresponding to the various labels on the diagram (12.3) are the appropriate instances of the hypothesis, claim, workspace and so on for the rule in question.

$$\begin{tikzcd}
& \mathsf{hyp} \arrow[d, "\mathsf{claimcon}"] \\
\mathsf{claim} \arrow[ur, "\mathsf{verif}"] \arrow[r, "\mathsf{hypcon}"'] & \mathsf{wksp}
\end{tikzcd} \tag{12.3}$$

While some of these are done in detail some others are not. For our purposes, it is enough to prove that a codification as an actual factorization in $\mathsf{SynCat}\big[\textbf{FinProd}, \mathsf{Name}[F]\big]$ (as defined in Section 10.1) is possible. In general, this may be done in more than one way. Symmetry and reflexivity are treated separately. Transitivity, concretion, abstraction and substitutivity are all treated in Section 12.2.7, as they all are special instances of Example 8.3.1.

12.2.1 Reflexivity

The equational rule of inference is

$$\frac{}{e =_V e}$$

Translation as a construction Translated into the present context, as an instance of the rule of construction REF, this is represented as

$$\text{REF}\quad \frac{A \xrightarrow{f} B}{A \underset{f}{\overset{f}{\rightrightarrows}} B}$$

where

$$\begin{aligned} f &:= \mathsf{Arr}\,[e, V, \mathsf{Type}[e]] \\ A &:= \mathsf{Type}[V] \\ B &:= \mathsf{Type}[e] \end{aligned}$$

This concludes the first step.

Expression as actual factorization The corresponding actual factorization:

$$\begin{tikzcd} & (\mathsf{ar} \times \mathsf{ar})^{\langle f, f \rangle} \arrow[d, "\mathsf{Proj}[1]"] \\ \mathsf{ar}^f \arrow[ur, "\Delta"] \arrow[r, "\mathsf{Id}[\mathsf{ar}]"'] & \mathsf{ar}^f \end{tikzcd} \tag{12.4}$$

where ar is the object of arrows in the sketch for categories.

Note that one can also use $\mathsf{Proj}[2]$ as *claimcon*. This factorization actually occurs in $\mathsf{SynCat}[\textbf{Cat}, F]$ and is inherited by $\mathsf{SynCat}[\textbf{FinProd}, F]$. A similar remark is true of the constructions for symmetry and transitivity.

12.2.2 Symmetry

Although Chapter 8, we not use a rule of construction corresponding to symmetry, we shall record an actual factorization for this to facilitate later discussion (in this section) on proofs as actual factorizations. The rule in equational deduction is

$$\frac{e =_V e'}{e' =_V e}$$

We define

$$\begin{aligned} f &:= \mathsf{Arr}\,[e, V, \mathsf{Type}[e]] \\ f' &:= \mathsf{Arr}\,[e', V, \mathsf{Type}[e]] \end{aligned}$$

then the actual factorization is as exhibited below:

$$\begin{tikzcd}
 & (\mathsf{ar}\times\mathsf{ar})^{\langle g,f\rangle} \arrow[d, "{\langle\mathsf{Proj}[2],\mathsf{Proj}[1]\rangle}"] \\
(\mathsf{ar}\times\mathsf{ar})^{\langle f,g\rangle} \arrow[ur, "{\langle\mathsf{Proj}[2],\mathsf{Proj}[1]\rangle}"] \arrow[r, "{\mathsf{Id}[\mathsf{ar}]\times\mathsf{Id}[\mathsf{ar}]}"'] & (\mathsf{ar}\times\mathsf{ar})^{\langle f,g\rangle}
\end{tikzcd}$$

12.2.3 Transitivity

The equational rule of inference is

$$\frac{e =_V e' \quad e' =_V e''}{e =_V e''}$$

For the first step we define

$$\begin{aligned}
f : D \to C \; &:= \mathsf{Arr}\,[e, V, \mathsf{Type}[e]] \\
g : D \to C \; &:= \mathsf{Arr}\,[e', V, \mathsf{Type}[e]] \\
h : D \to C \; &:= \mathsf{Arr}\,[e'', V, \mathsf{Type}[e]]
\end{aligned}$$

Note that f, g, and h have the same domain and the same codomain as e, e', and e'' have the same type and as V is the same in each of the terms exhibited below:

$$\text{TRANS} \quad \frac{D \overset{f}{\underset{g}{\rightrightarrows}} C \qquad\qquad D \overset{g}{\underset{h}{\rightrightarrows}} C}{D \overset{f}{\underset{h}{\rightrightarrows}} C}$$

The corresponding actual factorization is provided in Section 12.2.7.

12.2.4 Concretion

In this case the equational inference rule reads

Given a set V of typed variables, $x \in V$ and an equation $e =_V e'$ such that $x \in V \setminus (\mathsf{VarSet}[e] \cup \mathsf{VarSet}[e'])$, and given that $\mathsf{Type}[x]$ is inhabited,

$$\frac{e =_V e'}{e =_{V\setminus\{x\}} e'}$$

We define $\tau := \mathsf{Type}[e] = \mathsf{Type}[e']$ and $\sigma := \mathsf{Type}[x]$, and

$$\begin{aligned} f : P \to \tau &:= \mathsf{Arr}\,[e, V, \tau] \\ f' : P \to \tau &:= \mathsf{Arr}\,[e', V, \tau] \\ g : Q \to \tau &:= \mathsf{Arr}\,[e, V \backslash \{x\}, \tau] \\ g' : Q \to \tau &:= \mathsf{Arr}\,[e', V \backslash \{x\}, \tau] \end{aligned}$$

Using Lemma 12.1.1, we may choose a map

$$h : \prod (V \setminus \{x\}) \to \prod V$$

such that

$$\begin{aligned} g &= f \circ h \\ g' &= f' \circ h \end{aligned}$$

Thus coded as arrows, the rule reads

$$\frac{f = f'}{f \circ h = f' \circ h}$$

12.2.5 Abstraction

In this case the equational rule of inference reads

Given a set of typed variables and $x \in \mathsf{Vbl}[S] \setminus V$,

$$\frac{e =_V e'}{e =_{V \cup \{x\}} e'}$$

We define $\tau := \mathsf{Type}[e] = \mathsf{Type}[e']$ and $\sigma := \mathsf{Type}[x]$, and

$$\begin{aligned} f : P \to \tau &:= \mathsf{Arr}\,[e, V, \tau] \\ f' : P \to \tau &:= \mathsf{Arr}\,[e', V, \tau] \\ g : Q \to \tau &:= \mathsf{Arr}\,[e, V \cup \{x\}, \tau] \\ g' : Q \to \tau &:= \mathsf{Arr}\,[e', V \cup \{x\}, \tau] \end{aligned}$$

Using Lemma 12.1.1, we may choose a map

$$h : \prod (V \cup \{x\}) \to V$$

such that $g = f \circ h$ and $g' = f' \circ h$. Thus coded as arrows the rule reads

$$\frac{f = f'}{f \circ h = f' \circ h}$$

12.2.6 Substitutivity

Given a set V of typed variables, $x \in V$, and expressions u and u' for which $\mathsf{Type}[x] = \mathsf{Type}[u] = \mathsf{Type}[u']$ and $\mathsf{Type}[e] = \mathsf{Type}[e'] = \tau$,

$$\frac{e =_V e' \quad u =_W u'}{e[x \leftarrow u] =_{V \setminus \{x\} \cup W} e'[x \leftarrow u']}$$

We already have the representations

$$\begin{aligned}
f &:= \mathsf{Arr}\,[e, V, \tau] = \mathsf{Sep}[e]\,\mathsf{Par}[e]\,\mathsf{Dia}\,[e, V, \tau] \\
f' &:= \mathsf{Arr}\,[e', V, \tau] = \mathsf{Sep}[e']\,\mathsf{Par}[e']\,\mathsf{Dia}[e', V, \tau] \\
g &:= \mathsf{Arr}\,[u, W, \mathsf{Type}[u]] = \mathsf{Sep}[u]\,\mathsf{Par}[u]\,\mathsf{Dia}\,[u, W, \mathsf{Type}[u]] \\
g' &:= \mathsf{Arr}\,[u', W, \mathsf{Type}[u]] = \mathsf{Sep}[u']\,\mathsf{Par}[u']\,\mathsf{Dia}\,[u', W, \mathsf{Type}[u]] \\
h &:= \mathsf{Arr}\big[e[x \leftarrow u], (V \setminus \{x\}) \cup W, \tau\big] \\
h' &:= \mathsf{Arr}\big[e'[x \leftarrow u'], (V \setminus \{x\}) \cup W, \tau\big]
\end{aligned}$$

In view of Lemma 11.3.1, we may choose arrows $A := \mathsf{Insert}[u, x, V, W]$ and $A' := \mathsf{Insert}[u', x, V, W]$ for which $h = f \circ A$ and $h' = f' \circ A'$. Note that the assumptions $e =_V e'$ and $u =_W u'$ are equivalent to assuming that $f = f'$ and $A = A'$. It follows that when coded in terms of arrows, the rule reads

$$\frac{f = f' \qquad A = A'}{f \circ A = f' \circ A'}$$

12.2.7 Transitivity, concretion, abstraction and substitutivity as actual factorizations

Transitivity may be viewed as a special case of Theorem 8.3.1 once equations are interpreted as commutative diagrams as shown in Diagram (12.5):

$$\begin{tikzcd}
D \arrow[r, "h"] \arrow[d, "\mathsf{Id}[D]"'] & C \arrow[d, "\mathsf{Id}[C]"] \\
D \arrow[r, "f"'] \arrow[ur, "g"] & C
\end{tikzcd} \tag{12.5}$$

The fact that the two triangles commute means that $h = g$ and $g = f$. That the outside square commutes means that $h = f$.

In view of Lemma 12.1.1, concretion and abstraction can be seen to be special cases of the following: For every pair of formally equal arrows $f, f' : D \to C$ and for every $h : E \to D$, $f \circ h$ and $f' \circ h$ are formally equal. This can also be realized as a special case of the commutativity of Diagram (8.3.1), with choices

as shown:

$$\begin{tikzcd} E \arrow[r, "h"] \arrow[d, "h"'] & D \arrow[d, "f"] \\ D \arrow[r, "g"'] \arrow[ur, "\mathsf{Id}[D]"] & C \end{tikzcd} \tag{12.6}$$

Particular choices for h yield concretion and abstraction.

In substitutivity, in view of Lemma 12.1.1, we have the following in terms of arrows: For every pair of formally equal arrows $f, f' : D \to C$, and for every pair of formally equal arrows $A, A' : E \to D$, $f \circ A$ and $f' \circ A'$ are formally equal. This is also a special case of Diagram (8.3.1) as shown below:

$$\begin{tikzcd} E \arrow[r, "A"] \arrow[d, "A'"'] & D \arrow[d, "f"] \\ D \arrow[r, "f'"'] \arrow[ur, "\mathsf{Id}[D]"] & C \end{tikzcd} \tag{12.7}$$

On the basis of the preceding discussion we conclude that we may make choices for all nodes and arrows in the diagram

$$\begin{tikzcd} & \mathsf{\mathit{hyp}} \arrow[d, "\mathsf{\mathit{claimcon}}"] \\ \mathsf{\mathit{claim}} \arrow[ur, "\mathsf{\mathit{verif}}"] \arrow[r, "\mathsf{\mathit{hypcon}}"'] & \mathsf{\mathit{wksp}} \end{tikzcd}$$

so that the actual factorization in $\mathsf{SynCat}[\mathbf{\mathsf{FinProd}}, \mathsf{F}]$ codes transitivity, concretion, abstraction and substitutivity respectively.

12.3 Deductions as factorizations

We now show that deductions in MSEL (Section 9.5) correspond to actual factorizations in $\mathsf{SynCat}[\mathbf{\mathsf{FinProd}}, \mathsf{Name}[\mathsf{F}]]$.

We first need two lemmas.

12.3.1 Lemma *Given two actual factorizations in any syntactic category*

$$
\begin{tikzcd}
 & B \arrow[d, "c_1"] \\
H \arrow[ur, "u_1"] \arrow[r, "h_1"'] & W_1
\end{tikzcd}
\qquad
\begin{tikzcd}
 & C \arrow[d, "c_2"] \\
B \arrow[ur, "u_2"] \arrow[r, "h_2"'] & W_2
\end{tikzcd}
\tag{12.8}
$$

there is a node W and arrows c and h for which

$$
\begin{tikzcd}
 & C \arrow[d, "c"] \\
H \arrow[ur, "u_2 \circ u_1"] \arrow[r, "h"'] & W
\end{tikzcd}
\tag{12.9}
$$

is an actual factorization.

Proof As every node in a syntactic category (or $\mathsf{SynCat}[\mathsf{E}, \mathsf{F}]$) is the vertex of a limit cone over some diagram in E, we may choose

$$
\begin{aligned}
\Delta_1 &= \mathsf{BsDiag}[W_1] \\
\Delta_2 &= \mathsf{BsDiag}[W_2] \\
\Delta_B &= \mathsf{BsDiag}[B]
\end{aligned}
$$

to get the following in the category of diagrams of E:

$$
\begin{tikzcd}
 & \Delta_1 \arrow[d, "\alpha_1"] \\
\Delta_2 \arrow[r, "\alpha_2"'] & \Delta_B
\end{tikzcd}
$$

where α_1 and α_2 are the morphisms of diagrams that give rise to c_1 and h_2. As the category of diagrams in a category is small complete, we may form the pullback as shown:

$$
\begin{tikzcd}
\Delta \arrow[r, "\beta_1"] \arrow[d, "\beta_2"'] & \Delta_1 \arrow[d, "\alpha_1"] \\
\Delta_2 \arrow[r, "\alpha_2"'] & \Delta_B
\end{tikzcd}
\tag{12.10}
$$

Taking the limit over the diagrams corresponding to the vertices in (12.10) and using the lemmas in Chapter 5, we get the following diagram in $\mathsf{SynCat}[E, F]$:

$$\begin{tikzcd}
B \arrow[r, "h_2"] \arrow[d, "c_1"'] & W_2 \arrow[d, "d_2"] \\
W_1 \arrow[r, "d_1"'] & W
\end{tikzcd}$$

This gives the following diagram in $\mathsf{SynCat}[E, F]$:

$$\begin{tikzcd}
 & & C \arrow[d, "c_2"] \\
 & B \arrow[ur, "u_2"] \arrow[r, "c_2"'] \arrow[d, "c_1"] & W_2 \arrow[d, "d_2"] \\
H \arrow[ur, "u_1"] \arrow[r, "h_1"'] & W_1 \arrow[r, "d_1"'] & W
\end{tikzcd}$$

The lemma follows by setting $c := d_2 \circ c_2$ and $h := d_1 \circ h_1$.□

12.3.2 Definition The Diagram (12.9) is said to be obtained by **chaining** the diagrams in (12.8).

12.3.3 Remark Chaining is an analogue of getting the deduction $\frac{E_1}{E}$ given the deduction $\frac{E_1}{E_2}$ and $\frac{E_2}{E}$.

12.3.4 Lemma *Given two actual factorizations in any syntactic category*

$$\begin{tikzcd}
 & C_1 \arrow[d, "c_1"] \\
H_1 \arrow[ur, "u_1"] \arrow[r, "h_1"'] & W_1
\end{tikzcd}
\qquad
\begin{tikzcd}
 & C_2 \arrow[d, "c_2"] \\
H_2 \arrow[ur, "u_2"] \arrow[r, "h_2"'] & W_2
\end{tikzcd}$$

we have the factorization

$$\begin{tikzcd} & C_1 \times C_2 \arrow[d, "c_1 \times c_2"] \\ H_1 \times H_2 \arrow[ur, "u_1 \times u_2"] \arrow[r, "h_1 \times h_2"'] & W_1 \times W_2 \end{tikzcd}$$

Proof Omitted.□

12.4 From deduction to factorization

[To do: ***Restore the construction of a normal form for equational deductions that appeared in [Bagchi and Wells, 1996].*****]**

Now suppose we have a deduction D of an equation E from a list $P := (E_1, \ldots, E_n)$ of equations. We show how to give a factorization corresponding to the deduction for each of the four parts of Definition 9.5.1 of deduction.

D.1 $D = (E)$ and $P = (E)$. Let n be the node of $\mathsf{SynCat}[\textbf{\textsf{FinProd}}, \mathsf{F}]$ corresponding to E. Then the factorization is

$$\begin{tikzcd} & \mathsf{n} \arrow[d, "\mathsf{Id}[\mathsf{n}]"] \\ \mathsf{n} \arrow[ur, "\mathsf{Id}[\mathsf{n}]"] \arrow[r, "\mathsf{Id}[\mathsf{n}]"'] & \mathsf{n} \end{tikzcd}$$

D.2 $D = (E)$ and P is the empty list. Then E is $e =_V e$ and the factorization is given in Diagram (12.4), where $f = \mathsf{Arr}[e, V, \mathsf{Type}[e]$.

D.3 $D = (E, D_1)$, where D_1 is a deduction of an equation E_1 from P and

$$\frac{E_1}{E}$$

is an instance of a rule of inference R of MSEL.

In this case we have constructed the factorization for R in Section 12.2, and there is a factorization for D_1 by induction hypothesis. These may be chained (Lemma 12.3.1) to get the factorization for D.

D.4 $D = (E, D_1, D_2)$, where for $i = 1, 2$, D_i is a deduction of an equation E_i from a list of premises P_i, $P = P_1P_2$, and

$$\frac{E_1 \quad E_2}{E}$$

is an instance of a rule of inference R of MSEL.

By induction hypothesis, there are factorizations F_1 and F_2 for D_1 and D_2, and we have constructed a factorization F for R in Section 12.2. F_1 and F_2 may be combined into a single factorization by Lemma 12.3.4, and the resulting factorization may be chained with F to obtain a factorization for D.

12.5 From factorization to deduction

[**To do:** ***Write this section.***]

Chapter 13

Future work

13.1 More explicit rules of construction

It is noteworthy that the rules of construction for constructor spaces given in Section 4.4 correspond to arrows of FinLim, although not in a one-to-one way (see Remark 4.4.3). The rules are given here in a form that requires pattern recognition (recall the discussion in 8.5.4), but they clearly could be given at another level of abstraction as arrows or families of arrows of FinLim.

13.2 Comparison with Makkai's work

M. Makkai [1993a], [1993b] has produced an approach to explicating the logic of sketches that is quite different from that presented here. Both are attempts at codifying the process of diagram-manipulation used to prove results valid in particular structured categories. In both cases, structured categories or doctrines form the semantic universes with a pre-existing notion of validity. Both approaches are motivated by the desire to formulate a syntactic notion of deducibility. After that is said, the two approaches are very different and a detailed investigation of the relationship between them is desirable.

The following points are however worth mentioning.

- Both approaches require a generalization of Ehresmann sketches: Forms as in Definition 6.4 here and sketch category over a category G in [Makkai, 1993b].
- In Makkai's approach, the sketch axioms are different for different doctrines and serve both as axioms and as rules of inference. In contrast, here the rules of construction are the *same* for all doctrines; what distinguishes doctrines is their specification as CS-sketches. This feature is a departure from the usual practice in symbolic logic.

13.3 Equivalences with other logics

In Chapter 10, we worked out the details of the equivalence of multisorted equational logic and $\mathsf{FPTh}\big[Sk[\mathcal{S}]\big]$. The method used there should work for any logical system that can be described as a constructor-space sketch. Thus, in general, we shall have some logical system L and a category $\mathsf{CatTh}[E_L, F]$ in which E_L is the kind of category in which the models of L are. For instance, if L is the typed λ-calculus, E_L would be CCC, and if L is intuitionistic type theory, then E_L would be a constructor space for toposes.

Given any sound and complete deductive system for L, if we interpret terms as arrows and encode them in $\mathsf{CatTh}[E_L, F]$ as we have done here, then we conjecture that the method will show that all theorems of L can be realized

as actual factorizations in $\mathsf{CatTh}[\mathsf{E}_L, \mathsf{F}]$. (Indeed it appears nearly obvious that this will happen if we know that L and E_L have equivalent models; a detailed proof, is of course necessary to settle the matter.) In the examples of the preceding paragraphs, we might use the deductive systems formulated in [Lambek and Scott, 1986]. The method used here is quite general.

13.4 Syntax by other doctrines

The rules of construction given herein take place in the doctrine of finite limits. Most of the syntax and rules of deduction of string-based logical theories are clearly expressible using context-sensitive grammars, which intuitively at least can be modeled using finite limits. (Context-free grammars can be modeled using only finite products [Wells and Barr, 1988].) However, one could imagine extensions of this doctrine:

(i) Use coproducts to allow the specification of conditional compilation or other syntactical alternatives.

(ii) Use doctrines involving epimorphic covering families to allow the intentional description of ambiguous statements.

(iii) Use some extension of finite limit doctrines to allow the treatment of categories whose structure is determined only up to isomorphism (the usual approach in category theory) instead of being specified. There are two approaches in the literature: the use of categories enriched over groupoids and universal sketches as described in [Barr and Wells, 1992]. It is not clear that such methods are necessary for applications in computer science, but they may be for general applications to mathematical reasoning.

Bibliography

[Adámek and Rosicky, 1994] Jiři Adámek and Jiři Rosicky. **Locally Presentable and Accessible Categories**. Cambridge University Press, 1994.

[Bagchi and Wells, 1995] Atish Bagchi and Charles Wells. *Graph-based logic and sketches I: The general framework*. MSRI Preprint #1995-006, `http://www.msri.org/communications/preprints/online/1995-006.html`, 1995.

[Bagchi and Wells, 1996] Atish Bagchi and Charles Wells. *Graph-based logic and sketches II: Finite product categories and equational logic*. MSRI Preprint #1996-092, `http://www.msri.org/communications/preprints/online/1996-092.html`, 1996.

[Barr, 1971] Michael Barr. *Exact categories*. In Barr et al. [1971].

[Barr *et al.*, 1971] Michael Barr, Pierre A. Grillet, and Donovan H. van Osdol, editors. **Exact categories and Categories of Sheaves**, volume 236 of *Springer Lecture Notes in Mathematics*. Springer Verlag, 1971.

[Barr and Wells, 1985] Michael Barr and Charles Wells. **Toposes, Triples and Theories**, volume 278 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, New York, 1985. A list of corrections and additions is maintained in [Barr and Wells, 1993].

[Barr and Wells, 1992] Michael Barr and Charles Wells. *On the limitations of sketches*. Canadian Mathematical Bulletin, **35**:287–294, 1992.

[Barr and Wells, 1993] Michael Barr and Charles Wells. *Corrections to Toposes, Triples and Theories*. Available at `http://www.cwru.edu/CWRU/Dept/Artsci/math/wells/pub/papers.html`, 1993. Corrections and additions to [Barr and Wells, 1985].

[Barr and Wells, 1999] Michael Barr and Charles Wells. **Category Theory for Computing Science, third edition**. Les Publications CRM, Montréal, 1999. Available for purchase from the website at `http://crm.umontreal.ca/pub/Ventess/CatalogueEng.html`.

[Bastiani and Ehresmann, 1972] Andrée Bastiani and Charles Ehresmann. *Categories of sketched structures*. Cahiers de Topologie et Géométrie Différentielle, **13**:104–213, 1972.

[Brookes and Mislove, 1997] Stephen Brookes and Michael W. Mislove, editors. **Mathematical Foundations of Programming Semantics, Thirteenth Annual Conference**. Elsevier, 1997.

[Burroni, 1970a] Albert Burroni. *Esquisses des catégories à limites et des quasi-topologies*. Esquisses Mathématiques, **5**, 1970.

[Burroni, 1970b] E. Burroni. *Catégories discretement structurées*. Esquisses Mathématiques, **5**, 1970.

[Coppey and Lair, 1988] L. Coppey and C. Lair. *Leçons de théorie des esquisses, partie II*. Diagrammes, **19**, 1988.

[Duval and Reynaud, 1994a] Dominique Duval and Jean-Claude Reynaud. *Sketches and computation (1)*. Mathematical Structures in Computer Science, 1994.

[Duval and Reynaud, 1994b] Dominique Duval and Jean-Claude Reynaud. *Sketches and computation (2)*. Mathematical Structures in Computer Science, 1994.

[Duval and Sénéchaud, 1994] Dominique Duval and Pascale Sénéchaud. *Sketches and parametrization*. Theoretical Computer Science, 1994.

[Ebbinghaus *et al.*, 1984] H.-D. Ebbinghaus, J. Flum, and W. Thomas. **Mathematical Logic**. Springer-Verlag, 1984.

[Ehresmann, 1966] Charles Ehresmann. *Introduction to the theory of structured categories*. Technical Report 10, University of Kansas, 1966.

[Ehresmann, 1968a] Charles Ehresmann. *Catégories structurées généralisés*. Cahiers de Topologie et Géométrie Différentielle, **X**:139–168, 1968.

[Ehresmann, 1968b] Charles Ehresmann. *Esquisses et types des structures algébriques*. Bul. Inst. Polit. Iaşi, **XIV**, 1968.

[Eilenberg and Mac Lane, 1945] Samuel Eilenberg and Saunders Mac Lane. *General theory of natural equivalences*. Trans. Amer. Math. Soc., **58**:231–244, 1945.

[Fourman and Vickers, 1986] Michael Fourman and S. Vickers. *Theories as categories*. In **Category Theory and Computer Programming**, D. Pitt et al., editors, volume 240 of *Lecture Notes in Computer Science*, pages 434–448. Springer-Verlag, 1986.

[Goguen and Burstall, 1986] Joseph A. Goguen and R. M. Burstall. *A study in the foundations of programming methodology: Specifications, institutions, charters and parchments*. In **Category Theory and Computer Programming**, D. Pitt et al., editors, volume 240 of *Lecture Notes in Computer Science*, pages 313–333. Springer-Verlag, 1986.

[Goguen and Meseguer, 1982] Joseph A. Goguen and José Meseguer. *Completeness of many-sorted equational logic*. Technical Report CSL-135, SRI International Computer Science Laboratory, 333 Ravenswood Ave., Menlo Park, CA 94025, USA, 1982.

[Gray, 1987] J. W. Gray. *Categorical aspects of data type constructors.* Theoretical Computer Science, **50**:103–135, 1987.

[Gray, 1989] J. W. Gray. *The category of sketches as a model for algebraic semantics.* In **Categories in Computer Science and Logic**, John W. Gray and Andre Scedrov, editors, volume 92 of *Contemporary Mathematics*, pages 109–135. American Mathematical Society, 1989.

[Gray, 1990] John W. Gray. *Executable specifications for data-type constructors.* Diagrammes, **24**:7–31, 1990.

[Guitart, 1974] René Guitart. *Remarques sur les machines et les structures.* Cahiers de Topologie et Géométrie Différentielle Catégorique, **15**, 1974.

[Guitart and Lair, 1980] René Guitart and Christian Lair. *Calcul syntaxique des modèles et calcul des formules internes.* Diagrammes, **4**, 1980.

[Guitart and Van den Bril, 1977] René Guitart and Luc Van den Bril. *Decompositions et lax-completions.* Cahiers de Topologie et Géométrie Différentielle Catégorique, **18**, 1977.

[Kinoshita *et al.*, 1997] Yoshiki Kinoshita, John Power, and Makoto Takeyama. *Sketches.* In Brookes and Mislove [1997].

[Kock, 1967] Anders Kock. *Limit monads in categories.* Aarhus Preprint Series, 1967.

[Lair, 1987] Christian Lair. *Trames et semantiques catégoriques des systèmes de trames.* Diagrammes, **18**, 1987.

[Lambek and Scott, 1984] Joachim Lambek and P. Scott. *Aspects of higher order categorical logic.* In **Mathematical Applications of Category Theory**, J.W. Gray, editor, volume 30 of *Contemporary Mathematics*, pages 145–174. American Mathematical Society, 1984.

[Lambek and Scott, 1986] Joachim Lambek and P. Scott. **Introduction to Higher Order Categorical Logic**, volume 7 of *Cambridge Studies in Advanced Mathematics.* Cambridge University Press, 1986.

[Makkai, 1993a] Michael Makkai. *Generalized sketches as a framework for completeness theorems*, 1993. Journal of Pure and Applied Algebra, to appear.

[Makkai, 1993b] Michael Makkai. *The syntax of categorical logic*, 1993. Preprint, McGill University.

[Makkai and Paré, 1990] Michael Makkai and R. Paré. **Accessible Categories: the Foundations of Categorical Model Theory**, volume 104 of *Contemporary Mathematics.* American Mathematical Society, 1990.

[Makkai and Reyes, 1977] Michael Makkai and Gonzalo Reyes. **First Order Categorical Logic**, volume 611 of *Lecture Notes in Mathematics*. Springer-Verlag, 1977.

[McDonald and Stone, 1984] John L. McDonald and Arthur Stone. *Topoi over graphs*. Cahiers de Topologie et Géométrie Différentielle, **XXV**, 1984.

[Meseguer, 1989] José Meseguer. *General logics*. Technical Report CSL-89-5, SRI International Computer Science Laboratory, 333 Ravenswood Ave., Menlo Park, CA 94025, USA, 1989.

[Power and Wells, 1992] A.J. Power and Charles Wells. *A formalism for the specification of essentially algebraic structures in 2-categories*. Mathematical Structures in Computer Science, **2**:1–28, 1992.

[Tholen and Tozzi, 1989] Walter Tholen and Anna Tozzi. *Completions of categories and initial completions*. Cahiers de Topologie et Géométrie Différentielle Catégorique, pages 127–156, 1989.

[Wells, 1990] Charles Wells. *A generalization of the concept of sketch*. Theoretical Computer Science, **70**:159–178, 1990.

[Wells and Barr, 1988] Charles Wells and M. Barr. *The formal description of data types using sketches*. In **Mathematical Foundations of Programming Language Semantics**, Michael Main et al., editors, volume 298 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

Atish Bagchi
Department of Mathematics
Community College of Philadelphia
1700 Spring Garden St.
Philadelphia, PA 19130, USA
`atish@math.upenn.edu`

Charles Wells
Professor Emeritus of Mathematics
Case Western Reserve University
Current address:
1440 Randolph Avenue
Apt 323
Saint Paul, MN 55105, USA
`charles@abstractmath.org`