A CATEGORY-THEORETIC APPROACH TO THE
SEMANTICS OF PROGRAMMING LANGUAGES

Frank J. Oles
August 1982
I hereby grant permission for my thesis to be reproduced for noncommercial purposes.

Anyone who wishes to produce copies of any part or all of this thesis either for sale or for incorporation into a work for sale must first obtain a license from me.

Frank J. Oles, Ph.D. (Frankjosepholes@aol.com)  
February 4, 2012
A CATEGORY-THEORETIC APPROACH TO THE
SEMANTICS OF PROGRAMMING LANGUAGES

by
FRANK JOSEPH OLES

B. S., Case Western Reserve University, 1968
M. S., Cornell University, 1970

ABSTRACT OF DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer and
Information Science in the Graduate School of
Syracuse University
August, 1982
ABSTRACT OF DISSERTATION

Here we create a framework for handling the semantics of fully typed programming languages with imperative features, higher-order ALGOL-like procedures, block structure, and implicit conversions. Our approach involves the introduction of a new family of programming languages, the coercive typed \( \lambda \)-calculi, denoted by \( L \) in the body of the dissertation. By appropriately choosing the linguistic constants (i.e. generators) of \( L \), we can view phrases of variants of ALGOL as syntactically sugared phrases of \( L \).

This dissertation breaks into three parts. In the first part, consisting of the first chapter, we supply basic definitions and motivate the idea that functor categories arise naturally in the explanation of block structure and stack discipline. The second part, consisting of the next three chapters, is dedicated to the general theory of the semantics of the coercive typed \( \lambda \)-calculus; the interplay between posets, algebras, and Cartesian closed categories is particularly intense here. The remaining four chapters make up the final part, in which we apply the general theory to give both direct and continuation semantics.
for desugared variants of ALGOL. To do so, it is necessary to show certain functor categories are Cartesian closed and to describe a category $\Sigma$ of store shapes. An interesting novelty in the presentation of continuation semantics is the view that commands form a procedural, rather than a primitive, phrase type.
A CATEGORY-THEORETIC APPROACH TO THE
SEMANTICS OF PROGRAMMING LANGUAGES

by

FRANK JOSEPH OLSE

B. S., Case Western Reserve University, 1968
M. S., Cornell University, 1970

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer and
Information Science in the Graduate School of
Syracuse University
August, 1982
Copyright 1982

FRANK JOSEPH OLES
PREFACE

The aim of this work has been to explain the semantics of fully typed languages that combine imperative capabilities with ALGOL-like procedures and block structure. In particular this approach is meant to handle in an intuitively satisfying way the connections among language features such as higher-order types, implicit conversions, identifier binding, and stack discipline. At the core of this dissertation are the following six doctrines.

1. The collection of phrase types for a programming language is a highly structured mathematical object whose definition is dominated by the interaction between procedural types and implicit conversions. It is a free type algebra generated by the poset of primitive phrase types. (A type algebra has a category rather than a set as its underlying object, and its structure reflects the fact that hom functors are contravariant in their first argument and covariant in their second.)

2. By the device of incorporating the identifiers into the signature, a desugared, fully-typed version of ALGOL 60 is best viewed, not as a free algebra generated by its identifiers, but rather as a free algebra generated by what one usually views as operators (e.g., true, or, 0, 1, +, :=, etc.).
3. To check a program for mismatched types is to evaluate the application of a homomorphism to the program.

4. The meaning of a program or a program fragment, in the context of an assignment of types to its free identifiers, is a morphism in a Cartesian closed category.

5. Variable declarations should be viewed as changing, not the store of the machine, but rather the set of stores with respect to which commands are interpreted; the attempt to capture this intuition precisely leads to the natural introduction of Cartesian closed categories with non-functional morphisms.

6. The definition of functions by structural induction represents a failure to discern underlying algebraic structure; in so far as possible a syntactic or semantic function should be described as the unique morphism which satisfies some universal mapping property.

The debt I owe my advisor, Prof. John C. Reynolds, is substantial. His ideas, described in part in [6], [7], and [8], have had a profound impact on this work. In particular, the original suggestion that certain Cartesian closed categories whose objects are functors have a role to play in explaining stack discipline was his. Also, he drew my attention to the distinction between data types and phrase types in the course of discussing his ideas on the relevance of posets to the proper explanation of coercion. I would like also to thank Prof. F. Lockwood Morris
and Prof. Luis Sanchis. Special thanks go to A.V. Jategaonkar.

This dissertation was typed on I.B.M. Selectric type-
writers using five different kinds of type balls. Obviously,
the typists, Virginia Vedder, Barbara Merchant, and Bert
Fancher, deserve my eternal gratitude for putting this dis-
sertation into a readable form.

This material is based upon work partially supported by
National Science Foundation Grant MCS-8017577.
# CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PREFACE</td>
<td>iii</td>
</tr>
<tr>
<td>I.</td>
<td>THE UTILITY OF FUNCTOR CATEGORIES</td>
<td>1</td>
</tr>
<tr>
<td>II.</td>
<td>TYPE ALGEBRAS</td>
<td>28</td>
</tr>
<tr>
<td>III.</td>
<td>SYNTAX, SUBSTITUTION, AND TYPE CHECKING</td>
<td>50</td>
</tr>
<tr>
<td>IV.</td>
<td>THE FUNDAMENTAL THEOREM OF SEMANTICS</td>
<td>86</td>
</tr>
<tr>
<td>V.</td>
<td>CARTESIAN CLOSED CATEGORIES</td>
<td>138</td>
</tr>
<tr>
<td>VI.</td>
<td>A CATEGORY OF STORE SHAPES</td>
<td>162</td>
</tr>
<tr>
<td>VII.</td>
<td>DESUGARED ALGOL: DIRECT SEMANTICS</td>
<td>171</td>
</tr>
<tr>
<td>VIII.</td>
<td>DESUGARED ALGOL: CONTINUATION SEMANTICS</td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>BIBLIOGRAPHY</td>
<td>239</td>
</tr>
</tbody>
</table>
CHAPTER I
THE UTILITY OF FUNCTOR CATEGORIES

This chapter is intended to serve as a foundation upon which the rest of the dissertation rests. The foundational material is of two sorts. First, there are the definitions of terms and the descriptions of notations, familiarity with which is essential to understanding the subsequent work. Second, there is a comparatively informal explanation of the connection between functor categories and the semantics of block structure. The rest of the dissertation is meant (1) to provide the mathematical substance associated with this informal connection, (2) to incorporate at the same time intuitions about implicit conversions, type checking, and the algebraic nature of syntax and semantics, and (3) to explicitly describe the semantics of a desugared variant of an ALGOL-like language.

We begin with some notational conventions. When $X$ and $Y$ are simply sets, then $X \rightarrow Y$ denotes the set of functions from $X$ to $Y$. We write

$$ f \in X \rightarrow Y ,$$

rather than

$$ f : X \rightarrow Y .$$

All the infixed arrows used in this work are right-associative. Thus $W \rightarrow X \rightarrow Y \rightarrow Z$ means $W \rightarrow (X \rightarrow (Y \rightarrow Z))$. (This also applies to the heavy arrow $\Rightarrow$, which will be defined later.)
We abhor unnecessary parentheses. Therefore for functional application we write \( f \, x \) rather than \( f(x) \). Functional application associates to the left, so that \( f \, g \, h \, x \) means \(((f \, g) \, h) \, x\). Also, application binds more tightly than anything else, so that \( F \, a \, \times \, G \, b \) should be read as \((F \, a) \, \times \, (G \, b)\). However, if a few extra parentheses really enhance readability, we shall throw them in.

Suppose \( f \in X \rightarrow Y \). We write \( \text{dom} \, f \) for \( X \), the domain of \( f \), and \( \text{cod} \, f \) for \( Y \), the codomain of \( f \).

The function 
\[
[f \mid a\!:\!b] \in X \cup \{a\} \rightarrow Y \cup \{b\}
\]
is given by
\[
[f \mid a\!:\!b] \, x = \begin{cases} 
fx & \text{if } x \neq a \\
b & \text{if } x = a ,
\end{cases}
\]
where \( x \in X \cup \{a\} \).

Depending on the context, \( X^* \) means either the set of all strings of elements of \( X \), or the set of all functions \( f \in F \rightarrow X \)
where \( F \) is a finite subset of some specified set. The notation \( X^\infty \) denotes the set of all functions from a specified infinite set to \( X \); the notation \( X^\infty \) always occurs in conjunction with \( X^* \).

Ordered tuples are enclosed in angle brackets, e.g.
\[
(y, z) \in Y \times Z.
\]
Also, if \( f \in X \to Y \) and \( g \in X \to Z \), then we use angle brackets to denote the function

\[
\langle f, g \rangle \in X \to Y \times Z
\]

such that

\[
\langle f, g \rangle x = \langle f x, g x \rangle \quad \text{for all} \quad x \in X.
\]

In order to establish notation we define some familiar category-theoretic concepts.

An ordered 6-tuple

\[
C = \langle \text{Ob } C, \text{Ar } C, \text{dom}_C, \text{cod}_C, \circ_C, 1_C \rangle
\]
is a **category** if

1. \( \text{Ob } C \) (the objects of \( C \)) and \( \text{Ar } C \) (the arrows of \( C \)) are collections,
2. \( \text{dom}_C, \text{cod}_C \in \text{Ar } C \to \text{Ob } C \),
3. \( \circ_C \in \text{Ar } C \times_{\text{Ob } C} \text{Ar } C \to \text{Ar } C \), where

\[
\text{Ar } C \times_{\text{Ob } C} \text{Ar } C = \{ (\alpha, \beta) \in \text{Ar } C \times \text{Ar } C \mid \text{dom}_C \alpha = \text{cod}_C \beta \}
\]

(as usual, call \( \circ_C \) composition and write \( \alpha \circ_C \beta \) rather than \( \circ_C (\alpha, \beta) \)),
4. \( \text{dom}_C (\alpha \circ_C \beta) = \text{dom}_C \beta, \text{cod}_C (\alpha \circ_C \beta) = \text{cod}_C \alpha \),

and composition is associative,
5. \( 1_C \in \text{Ob } C \to \text{Ar } C \), and
6. for all \( X \in \text{Ob } C \), \( \text{dom}_C (1_C X) = X = \text{cod}_C (1_C X) \),

and \( 1_C X \) is a two-sided identity for composition.
The subscript $C$ may be omitted if it is clear from context, and alternative notations for $l^C_X$ are $l^C_X$ and $l^C_X$. The names dom and cod abbreviate domain and codomain, respectively. Arrows are also called morphisms, and the set of arrows $\alpha$ from $X$ to $Y$, i.e., such that $\text{dom } \alpha = X$ and $\text{cod } \alpha = Y$, is denoted $X \longrightarrow Y$.

Set denotes the category of sets and functions.

A functor $F$ consists of a domain $C$ and a codomain $D$, both of which are categories, and functions $\text{Ob } F \in \text{Ob } C \longrightarrow \text{Ob } D$ and $\text{Ar } F \in \text{Ar } C \longrightarrow \text{Ar } D$ such that

1. if $\alpha \in X \longrightarrow Y$, then $\text{Ar } F \alpha \in \text{Ob } F X \longrightarrow \text{Ob } F Y$,
2. if $\alpha \in X \longrightarrow Y$ and $\beta \in Y \longrightarrow Z$, then $\text{Ar } F (\beta \circ \alpha) = (\text{Ar } F \beta) \circ (\text{Ar } F \alpha)$, and
3. if $X \in \text{Ob } C$, then $\text{Ar } F (l^C_X) = l^D_{\text{Ob } F X}$

The collection of functors with domain $C$ and codomain $D$ is denoted $C \rightarrow D$. Following usual mathematical practice, we write simply $F$ rather than $\text{Ob } F$ or $\text{Ar } F$ unless there is a need to be exceptionally precise. The most noteworthy functor in $C \rightarrow C$ is the identity functor, which is denoted $l^C_C$ and which acts as the identity function on objects and arrows. Of course, functors may be composed in the usual boring way so as to obtain $G \circ F \in C \rightarrow E$ from $F \in C \rightarrow D$ and $G \in D \rightarrow E$.

A natural transformation $\eta$ consists of a domain $F$ and a codomain $G$, both of which are functors with identical domains
and identical codomains (say, \( F, G \in \mathcal{C} \to \mathcal{D} \)), and an 
\((\text{Ob} \ \mathcal{C})\)-indexed collection of arrows of \( \mathcal{D} \)

\[ \eta_X : FX \to GX, \text{ for } X \in \text{Ob} \ \mathcal{C} \]

such that, for all \( \alpha \in X \to Y \), the diagram

\[
\begin{array}{ccc}
FX & \xrightarrow{\eta_X} & GX \\
\downarrow F\alpha & & \downarrow G\alpha \\
FY & \xrightarrow{\eta_Y} & GY \\
\end{array}
\]

commutes in \( \mathcal{D} \).

Suppose \( \mathcal{C} \) and \( \mathcal{D} \) are categories. Of fundamental
importance in the sequel is the functor category \( \mathcal{C} \to \mathcal{D} \)
(alternative notation: \( \mathcal{D}^\mathcal{C} \)) whose objects are all functors
\( F \in \mathcal{C} \to \mathcal{D} \), and whose morphisms are all natural transformations
between functors in \( \mathcal{C} \to \mathcal{D} \). Composition in \( \mathcal{C} \to \mathcal{D} \) is defined
as follows: if \( \eta \in F \xrightarrow{\mathcal{C} \to \mathcal{D}} G \) and \( \nu \in G \xrightarrow{\mathcal{C} \to \mathcal{D}} H \), then
\( \nu \circ \eta \in F \xrightarrow{\mathcal{C} \to \mathcal{D}} H \) is given by

\[ (\nu \circ \eta)_X = (\nu_X) \circ (\eta_X) \quad \text{for all } X \in \text{Ob} \ \mathcal{C}. \]

We omit the details.

Each category \( \mathcal{C} \) has an opposite category denoted \( \mathcal{C}^{\text{op}} \)
and defined by

1. \( \text{Ob} \ \mathcal{C}^{\text{op}} = \text{Ob} \ \mathcal{C} \),
2. \( \text{Ar} \ \mathcal{C}^{\text{op}} = \text{Ar} \ \mathcal{C} \),
3. \( \text{dom}_{\mathcal{C}^{\text{op}}} = \text{cod}_\mathcal{C} \).
(4) \( \text{cod}_{\text{cop}} = \text{dom}_C \),

(5) for all composable pairs of arrows \( \alpha \) and \( \beta \),
\[
\alpha \circ_{\text{cop}} \beta = \beta \circ_C \alpha,
\]

(6) \( 1_{\text{cop}} = 1_C \).

To each functor \( F \in C \to \mathcal{D} \) there corresponds a functor \( F^{\text{op}} \in C^{\text{op}} \to \mathcal{D}^{\text{op}} \) which agrees with \( F \) on objects and arrows but has different domain and codomain.

If \( \langle C, \mathcal{D} \rangle \) is an ordered pair of categories, then its **product** \( C \times_{\text{Cat}} \mathcal{D} \), also denoted \( C \times \mathcal{D} \) when the abbreviated notation causes no confusion, is defined by

1. \( \text{Ob} (C \times \mathcal{D}) = (\text{Ob} C) \times (\text{Ob} \mathcal{D}) \),
2. \( \text{Ar} (C \times \mathcal{D}) = (\text{Ar} C) \times (\text{Ar} \mathcal{D}) \),
3. \( \text{dom}_{C \times \mathcal{D}} (\alpha, \beta) = (\text{dom}_C \alpha, \text{dom}_\mathcal{D} \beta) \),
4. \( \text{cod}_{C \times \mathcal{D}} (\alpha, \beta) = (\text{cod}_C \alpha, \text{cod}_\mathcal{D} \beta) \),
5. \( (\alpha', \beta') \circ_{C \times \mathcal{D}} (\alpha, \beta) = (\alpha' \circ_C \alpha, \beta' \circ_\mathcal{D} \beta) \), and
6. \( 1_{C \times \mathcal{D}} (X, Y) = (1_C X, 1_\mathcal{D} Y) \).

Each pair of functors \( F \in C \to C' \) and \( G \in \mathcal{D} \to \mathcal{D}' \) determines a product functor
\[
F \times G \in C \times C' \to \mathcal{D} \times \mathcal{D}'
\]
which acts on objects and arrows by applying \( F \) to the first component and \( G \) to the second.

Let \( C \) be a category, and let \( I \) be a set. We say \( C \) has **I-ary products** if for each \( I \)-indexed collection
\[
X \in I \to \text{Ob} C
\]
of objects of \( C \) there is some product diagram satisfying the usual universal mapping property, i.e. there is an
object $C$ and an $I$-indexed collection

$$\pi \in I \rightarrow \text{Ar } C, \text{ where } \pi \in C \rightarrow X_i \text{ for all } i \in I$$

of arrows (called projections) of $C$ such that

$$D \in \text{Ob } C, \alpha \in I \rightarrow \text{Ar } C, \text{ and } \alpha_i \in D \rightarrow X_i \text{ for all } i \in I$$

implies the existence of a unique

$$\hat{\alpha} \in D \rightarrow C$$

making the diagrams

$$\begin{array}{c}
\text{D} \\
\hat{\alpha} \\
\downarrow \alpha_i \\
\text{C} \quad \pi_i \rightarrow X_i
\end{array}$$

commute for all $i \in I$. Of course, a category with $I$-ary products may have more than one product diagram for an $I$-indexed collection of objects. Our interest is mainly in categories having $I$-ary products specifically given. In this situation, the object in the distinguished product diagram for $X \in I \rightarrow \text{Ob } C$ is denoted $\prod_I X$, and the $I$-indexed collection of distinguished projections arrows is written

$$\text{proj}_I \in I \rightarrow \text{Ar } C,$$

so that for all $i \in I$

$$\text{proj}_I i \in \prod_I X \rightarrow X_i.$$

We say that $C$ has finite products specifically given if $C$ has $I$-ary products specifically given for each finite set $I$; in this instance $C$ has a distinguished terminal object.
\[ \Pi_{\phi} \text{emp}_C, \text{ where} \]
\[ \text{emp}_C \in \phi \to \text{Ob } C \]

denotes the empty collection.

Binary products are usually denoted with \( \times \). We extend the use of angle brackets from \( \mathcal{S}e\ell \) to an arbitrary category \( C \) with a distinguished binary product functor by letting \( \langle f, g \rangle \) denote the unique arrow such that

\[
\begin{array}{ccc}
X & \xrightarrow{f} & Y \\
\downarrow{g} & & \downarrow{\langle f, g \rangle} \\
Y & \xrightarrow{\text{proj}_Y} & Y \times Z \\
\downarrow{\text{proj}_Z} & & \downarrow{\text{proj}_Z} \\
Z & \xrightarrow{\text{proj}_Z} & Z
\end{array}
\]

commutes, where the horizontal arrows are projections.

A category \( K \) with finite products specifically given is a **Cartesian closed category** if the following three conditions hold:

(Cart1) \( K \) has a distinguished terminal object \( K_{\text{term}} \)

(note that we do not require \( K_{\text{term}} = \Pi_{\phi} \text{emp}_K \),

(Cart2) \( K \) has a distinguished binary product functor

\[ K_\times \in K \times K \to K \]

(notation: \( K_\times \langle X, Y \rangle = X \times_K Y = X \times Y \))

(Cart3) for each ordered pair \( Y, Z \in \text{Ob } K \) there is \( Y \to Z \in \text{Ob } K \) and

\[ \text{AP}_K \langle Y, Z \rangle \in (Y \to Z) \times Y \xrightarrow{K} Z \]
such that whenever \( X \in \text{Ob} \; K \) and

\[
\alpha \in X \times Y \xrightarrow{\_} Z
\]

there exists a unique arrow

\[
\text{Ab}_K(X, Y, Z) \alpha \in X \xrightarrow{\_} (Y = Z)
\]

making the diagram commute (notation:

\[
y \Rightarrow z = z^y = y \xrightarrow{\_} z = K_\Rightarrow \langle y, z \rangle
\].

It is not hard to see that, for a Cartesian closed category \( K \), there is a unique way to extend the definition of \( K_\Rightarrow \) to give a functor

\[
k_\Rightarrow \in \text{K}^\text{op} \times K \rightarrow K
\].

Indeed, if

\[
\beta \in Y \xrightarrow{\_} Y \quad \text{and} \quad \nu \in Z \xrightarrow{\_} Z
\]

then

\[
k_\Rightarrow \langle \beta, \nu \rangle = \beta \Rightarrow \nu
\]

is the unique arrow such that
commutes. The functor $k_\to$ is sometimes called an "internal" hom-functor because it is reminiscent of the "external" hom-functor

$$\text{hom}_k \in k^{\text{op}} \times k \to \text{Set}$$

When $X$, $Y$, and $Z$ are understood, we write

$$\text{Ab} \text{ or } \text{Ab}_k \text{ for } \text{Ab}_k (X, Y, Z) \text{ ("abstraction")},$$

and

$$\text{Ap} \text{ or } \text{Ap}_k \text{ for } \text{Ap}_k (Y, Z) \text{ ("application")}.$$

Of course, $\text{Set}$ is the most common Cartesian closed category, and for it the internal and external hom-functors coincide. Another easily verifiable fact about Cartesian closed categories is that $\text{Ap}$ is natural in its second argument. Thus, for $Y, Z, Z' \in \text{Ob } k$ and $\alpha \in Z \xrightarrow{k} Z'$, the diagram

$$\begin{array}{ccc}
(Y \to Z) \times Y & \xrightarrow{\text{Ap} (Y, Z)} & Z \\
(1 \to \alpha) \times 1 & \downarrow & \downarrow \alpha \\
(Y \to Z') \times Y & \xrightarrow{\text{Ap} (Y, Z')} & Z'
\end{array}$$

commutes. In IV.6 of MacLane [4] an equivalent definition of Cartesian closed category in terms of specified right adjoints may be found.
The point of the discussion so far has been to fix notation for some standard concepts in category theory. Definitions of other category-theoretic terms which appear in the sequel may be found in MacLane [4] or Arbib and Manes [1]. We now move on to posets.

A partial order on a set $X$ is a relation on $X$, i.e., a subset of $X \times X$, which is reflexive, antisymmetric, and transitive. An ordered pair

$$P = \langle \text{Ob } P, \text{Ar } P \rangle$$

is a poset if $\text{Ob } P$ is a set and $\text{Ar } P$ is a partial order on $\text{Ob } P$. The assertion $\langle x, y \rangle \in \text{Ar } P$ is usually written $x \preceq_P y$. In addition the expression $x \preceq_P y$ is also used to denote the ordered pair $\langle x, y \rangle$ when it is true that $\langle x, y \rangle \in \text{Ar } P$. The correct reading of $x \preceq_P y$ will be clear from context. The poset $P$ becomes a category if we make the canonical definitions:

1. $\text{dom}_P (x \preceq_P y) = x$,
2. $\text{cod}_P (x \preceq_P y) = y$,
3. $(y \preceq_P z) \circ (x \preceq_P y) = (x \preceq_P z)$, and
4. $1_P x = (x \preceq_P x)$.

For posets $P$ and $Q$, the object part of any functor

$$F : P \to Q$$

is a monotone function, i.e. $x \preceq_P y$ implies $F x \preceq_Q F y$. Conversely, any monotone function $F : \text{Ob } P \to \text{Ob } Q$ has a
unique extension to a functor \( F \in \mathcal{P} \to \mathcal{Q} \). The adjectives "faithful" and "full" in the context of posets have the following meanings. A functor is faithful if it is injective on objects. A functor \( F \in \mathcal{P} \to \mathcal{Q} \) is full if for all \( x, y \in \text{Ob} \mathcal{P} \),

\[
x \preceq_{\mathcal{P}} y \iff Fx \preceq_{\mathcal{Q}} Fy.
\]

Let \( \mathcal{R} \) be the poset whose objects are monotone functions from \( \mathcal{P} \) to \( \mathcal{Q} \) and whose partial order is given by

\[
f \preceq_{\mathcal{R}} g \iff fx \preceq_{\mathcal{Q}} gx \quad \text{for each } x \in \text{Ob} \mathcal{P}.
\]

We may compare the canonical category derived from \( \mathcal{R} \) with the functor category \( \mathcal{P} \to \mathcal{Q} \) created from the canonical categories derived from \( \mathcal{P} \) and \( \mathcal{Q} \). It is easy to see that they are isomorphic. Indeed, the clarity of the subsequent exposition is not harmed by writing \( \mathcal{R} = \mathcal{P} \to \mathcal{Q} \) and \( f \preceq_{\mathcal{P} \to \mathcal{Q}} g \) rather than \( f \preceq_{\mathcal{R}} g \).

A **predomain** is a directed-complete poset, i.e. a poset all of whose directed subsets have least upper bounds, and a **domain** is a predomain that has a minimal element. The minimal element of a domain \( \mathcal{P} \) is denoted \( \mathbf{1} \) (read "bottom") or \( 1_{\mathcal{P}} \). For predomains \( \mathcal{P} \) and \( \mathcal{Q} \), a function \( f \in \text{Ob} \mathcal{P} \to \text{Ob} \mathcal{Q} \) is said to be **continuous** if \( f \) preserves least upper bounds of directed subsets, i.e., for each directed subset \( D \) of \( \mathcal{P} \),

\[
f(\bigcup_{\mathcal{P}} D) = \bigcup_{\mathcal{Q}} (fD).
\]

Of course, compositions of continuous functions are continuous, as also are identity functions. We are thus led to \( P\text{dom} \), the category whose objects are
predomains and whose arrows are continuous functions, with composition in $P\text{dom}$ being functional composition. The category $\text{Dom}$ is defined to be the full subcategory of $P\text{dom}$ whose objects are domains. A continuous function $f \in P \xrightarrow{\text{dom}} Q$ between domains is strict if it is bottom-preserving, i.e. $f \bot P = \bot Q$. By taking as objects all domains and as morphisms all strict continuous functions, we obtain $S\text{dom}$, a subcategory of $\text{Dom}$. In Chapter V we will show that both $P\text{dom}$ and $\text{Dom}$ are Cartesian closed categories; for either of them the underlying set of $P \Rightarrow Q$ is the set of continuous functions from $P$ to $Q$.

From each predomain $P$ we may create a domain $P_+$ by simply adding an element not already in $P$ as a minimal element. In most contexts the added minimal element is thought of as an "undefined element of $P."$

The connection between $\text{Set}$ and $P\text{dom}$ is worth exploring. There is an obvious forgetful functor

$$U \in P\text{dom} \longrightarrow \text{Set}$$

that forgets partial orders and regards continuous functions simply as functions. For each set $S$, let $\leq_S$ be the discrete partial order on $S$. We then get an embedding functor.

$$E \in \text{Set} \longrightarrow P\text{dom}$$

given by

$$ES = \langle S, \leq_S \rangle \text{ for } S \in \text{Ob Set, } E\sigma = \sigma \text{ for } \sigma \in \text{Ar Set.}$$

Clearly, $U \circ E = 1_{\text{Set}}$. Also

$$S \xrightarrow{\text{Set}} UP = ES \xrightarrow{P\text{dom}} P$$

for all sets $S$ and all predomains $P$. Thus $E$ is a left adjoint
of $U$, and $E Set$ (the image of $E$) is a full subcategory of $P_{dom}$ that is isomorphic to $Set$. For completeness, we note that $E$ is also the right adjoint of the functor in $P_{dom} \rightarrow Set$ which assigns to each predomain its set of equivalence classes modulo the least equivalence relation containing the partial order. The proof is easy.

This completes our review of definitions and notations. We now embark on an informal attack on the nature of block structure.

An intuitive grasp of the nature of variable declarations and of what is happening as one enters and exits from blocks is essential to programming in an ALGOL-like language. However, a precise semantic description of the constructs involved is difficult, and it is particularly difficult if one wants semantics for block structure that mesh elegantly with semantics for procedures.

Our goal is to outline in general terms how functor categories can be used to explain the semantics of ALGOL-like languages which possess

1) a rich type structure,

2) higher-order procedures, whose types may be arbitrarily complex,

3) imperative capabilities,

and

4) block structure.

The principal intuition we will try to capture is that during the execution of a program written in a language that obeys a
stack discipline not only is the store changed by commands (statements), but also the shape of the store is changed by variable declarations during block entrances and by block exits.

An extended description of the characteristics of the kinds of languages in which we are interested can be found in [6]. We have not worked out the details of the semantics of every language feature mentioned there, but we do know how to deal with the most important features, and we can tell a complete story about block structure.

For our purposes we think of a language as beginning with the specification of a set \( \mathcal{D} \) of data types. Let's say \texttt{int} (for integer) and \texttt{bool} (for boolean) are elements of \( \mathcal{D} \). Along with \( \mathcal{D} \) we have in mind \texttt{Val}, an assignment of sets (of values) to data types. Say

\[
\text{Val} \texttt{int} = \mathbb{Z} \text{ (the set of integers)}
\]

and

\[
\text{Val} \texttt{bool} = \{\text{true, false}\}.
\]

Each data type \( \delta \in \mathcal{D} \) gives rise to three primitive phrase types.

1) \( \delta\text{-exp} \) (for \( \delta \)-expression),
2) \( \delta\text{-var} \) (for \( \delta \)-variable),
3) \( \delta\text{-acc} \) (for \( \delta \)-acceptor).

Another primitive phrase type is \texttt{comm} (for command). The reader probably has an excellent intuitive feeling for the nature of phrases that are assigned the phrase types \( \delta\text{-exp}, \delta\text{-var}, \) and \texttt{comm}. A phrase has type \( \delta\text{-acc} \) if it gobbles up
values of type $\delta$ and produces commands. Thus, if $x$ and $y$
are $\delta$-variables, then in the assignment command $x := y$ the
$\delta$-variable $y$ is used as a $\delta$-expression and the $\delta$-variable $x$
is used as a $\delta$-acceptor. (Hence, even if a language turns out
not actually to have any phrases of type $\delta$-acc, it is still
useful to introduce that phrase type in order to explain $\delta$-var.)
A complete exposition of the connections between $\delta$-exp, $\delta$-var,
and $\delta$-acc would involve us in implicit conversions, a topic
that we will take up in subsequent chapters. See also
Reynolds [8].

The collection of all phrase types, denoted by $T$, cer-
tainly contains the primitive phrase types already described.
Also, for all elements $\tau$, $\theta$ of $T$, there is a phrase type
$\tau \Rightarrow \theta$ that is assigned to procedures accepting arguments
(i.e. having formal parameters) of type $\tau$ and producing
results (i.e. having calls) of type $\theta$. The use of the same
symbol for both procedural phrase types and exponentiation
in a Cartesian closed category is, of course, intentional.

An integral part of the semantics of a typed language
is the assignment to each phrase type $\tau$ of a meaning, denoted
$\text{Mng } \tau$. Perhaps with the plan of arranging matters so that
program fragments of type $\tau$ denote elements of $\text{Mng } \tau$, one
might suppose $\text{Mng } \tau$ is a set. However, the possible exist-
ence of nonterminating programs, which lead to an "undefined"
state, provides an inducement to partially order $\text{Mng } \tau$, where
the relation $x \preceq y$ means $x$ is "more undefined" than $y$. For
instance, see [9]. Following this line of reasoning, a
command which never terminates, such as while true do noaction, denotes the minimal element in Mnq \texttt{comm}. The need to give meanings to recursively defined expressions of type \( \tau \) causes us to require that directed subsets of Mnq \( \tau \) have least upper bounds, i.e. that Mnq \( \tau \) is a predomain. (Also, we generally want Mnq \( \tau \) to have a minimal element, but we must tread cautiously at this point to avoid becoming overcommitted to the use of \( \text{Dom} \) rather than \( \mathcal{P}\text{dom} \). As we shall see later, \( \text{Dom} \) is technically inadequate.)

Suppose we try only to give the semantics of programs which do not contain block entrances and exits. We start by positing the existence of a set \( S \) of possible stores. Regard sets as discretely ordered predomains. Since a command is a transformation of \( S \) that possibly may not terminate, and a function from \( S \) to \( S_\perp \) is the same as a continuous function from \( S \) to \( S_\perp \), we expect

\[
\text{Mng \texttt{comm}} = S \Rightarrow S_\perp.
\]

Here \( \Rightarrow \) is the internal hom functor for \( \mathcal{P}\text{dom} \).

Also, for each data type \( \delta \),

\[
\begin{align*}
\text{Mng } \delta\text{-exp} &= S \Rightarrow (\text{Val } \delta)_\perp, \\
\text{Mng } \delta\text{-acc} &= \text{Val } \delta \Rightarrow \text{Mng comm}, \\
\text{Mng } \delta\text{-var} &= \text{Mng } \delta\text{-acc} \times \text{Mng } \delta\text{-exp}
\end{align*}
\]

In other words, a \( \delta \)-expression attempts to compute a value of type \( \delta \) from the current store, a \( \delta \)-acceptor uses a value of type \( \delta \) to update the current store, and a \( \delta \)-variable may be used as either a \( \delta \)-expression or a \( \delta \)-acceptor. Finally, for
all τ, θ ∈ T, we expect

\[ \text{Mng} (\tau \rightarrow \theta) = \text{Mng} \tau \rightarrow \text{Mng} \theta, \]

i.e. the predomain of meanings for the procedural phrase type τ → θ is the predomain of continuous functions from Mng τ to Mng θ.

Although the approach of the preceding paragraph is attractively comprehensible, it is inadequate for the semantics of block structure because the set S is fixed throughout the exposition. The whole point of block structure is to permit S to vary during program execution. For instance, if we view stores as being functions from finite sets of locations in memory to the set of data-type values, then the domains of those functions may be regarded as store shapes. Variable declarations at the start of a block alter the shape of the store by adding locations to it, whereas block exit restores the shape of the store to its condition at block entrance. The semantics of a language obeying a stack discipline should reflect this dynamic behavior.

Therefore, let Σ be the collection of all store shapes. To each X ∈ Σ, there is a set St X of stores of that shape. Since the meaning of τ ∈ T varies with the store shape, Mng τ is not a predomain, but is rather a Σ-indexed collection of predomains. For instance we might arrange matters so that

\[ \text{Mng comm X} = \text{St X} \rightarrow (\text{St X})_\perp, \]

\[ \text{Mng δ-exp X} = \text{St X} \rightarrow (\text{Val δ})_\perp, \]
Mng $\delta$-acc $X = \text{Val } \delta \Rightarrow \text{Mng } \text{comm } X,$

$\text{Mng } \delta$-var $X = \text{Mng } \delta$-exp $X \times \text{Mng } \delta$-acc $X,$

where $X \in \Sigma, \delta \in D.$

It is important to realize that for $\tau \in \mathcal{T}$ and $X, Y \in \Sigma$ the predomains $\text{Mng } \tau X$ and $\text{Mng } \tau Y$ cannot be arbitrarily different. After all, we want the notion of command to have a uniform meaning for all store shapes or else the definition of operations like $;$ (concatenation of commands) will be bizarrely complicated. For instance, consider the program skeleton

\begin{verbatim}
begin int-var x;
  
  .
  .
  x := 3;
  .
  .

begin bool-var y;
  
  .
  .
  x := 3;
  .
  .

end;
  
  .
  .

end
\end{verbatim}

Suppose $X$ is the store shape corresponding to the outer block and $Y$ is the store shape corresponding to the inner block. Then $\text{Mng } \text{comm } X$ is relevant to the first occurrence of the assignment command $x := 3,$ while $\text{Mng } \text{comm } Y$ is relevant to the second occurrence. However, both occurrences are meant
to alter the contents of the same location. Roughly speaking, the fact that $X$ can be "expanded" to give $Y$ induces a function from $\text{Mng} \ comm \ X$ to $\text{Mng} \ comm \ Y$. So it becomes important to contemplate the notion of an expansion from a store shape $X$ to a store shape $Y$. Certainly, expansions ought to be composable. The composition ought to be associative. For each store shape $X$, there ought to be an identity expansion which involves "doing nothing" to $X$. In short, we assert that we erred in letting $\Sigma$ be the collection of store shapes. From now on, take $\Sigma$ to be the category of store shapes. The morphisms of $\Sigma$ are called expansions. Furthermore, for each phrase type $\tau$ we should require that $\text{Mng} \ \tau$ be a functor

\[
\text{Mng} \ \tau : \Sigma \longrightarrow \text{Pdom};
\]

this will elegantly take care of how an expansion

\[
\sigma : X \xrightarrow{\Sigma} Y
\]

induces a function

\[
\text{Mng} \ \tau \circ \sigma : \text{Mng} \ \tau \ X \longrightarrow \text{Pdom} \longrightarrow \text{Mng} \ \tau \ Y.
\]

Procedural phrase types are a bit tricky. Let $\tau, \theta \in \mathcal{T}$. Recall that in the simpler setting the predomain of meanings of type $\tau = \theta$ was the set of continuous functions from $\text{Mng} \ \tau$ to $\text{Mng} \ \theta$. One might hope that in the more sophisticated setting, where $\text{Mng} (\tau = \theta)$ is to be a functor, that the set of procedural meanings of type $\tau = \theta$ in the context of a store shape $X$ would be the set of continuous functions from $\text{Mng} \ \tau \ X$ to $\text{Mng} \ \theta \ X$, i.e.
\[ \text{Mng} \ (\tau \to \theta) \ X = \text{Mng} \ \tau \ X \to \text{Mng} \ \theta \ X. \]

Alas, this does not define a functor because \( \Rightarrow \) in \( \text{Pdom} \) is contravariant in its first argument. (For the same reason, one cannot define a functor from \( \text{Set} \) to \( \text{Set} \) by diagonalizing the hom functor, i.e. by letting \( X \) go to \( X \to X \).) Another idea might be to recall that \( \text{Mng} \ \tau \) and \( \text{Mng} \ \theta \) are objects of a functor category and to try letting \( \text{Mng} \ (\tau \Rightarrow \theta) \) be the set of natural transformations from \( \text{Mng} \ \tau \) to \( \text{Mng} \ \theta \). That's plain nonsense, because there is no way to regard such a set of natural transformations as a functor. We are, however, getting closer to the heart of the matter. In Chapter V we will prove that the functor category \( \Sigma \Rightarrow \text{Pdom} \) is Cartesian closed. Therefore, the appropriate equation governing meanings of procedural types is just

\[ \text{Mng} \ (\tau \Rightarrow \theta) = \text{Mng} \ \tau \Rightarrow \text{Mng} \ \theta, \]

where the heavy arrow on the right is exponentiation in the functor category \( \Sigma \Rightarrow \text{Pdom} \). Further evidence of the essential rightness of this equation will be presented at the conclusion of this section.

The reader may wonder why we didn't engineer this discussion so as to end up with the functor category \( \Sigma \Rightarrow \text{Dom} \). Unfortunately, contrary to the claim in [6], it does not appear that \( \Sigma \Rightarrow \text{Dom} \) is a Cartesian closed category, in spite of the fact that \( \text{Dom} \) is Cartesian closed.

This all sounds nice enough, but we have lost somewhere the idea that phrases of type \( \tau \) should have denotations
which are elements of \( \text{Mng} \, \tau \), because \( \text{Mng} \, \tau \) doesn't seem to have any elements if it is a functor. The solution is to reject even in the simple setting where block structure is ignored the intuition that a phrase of type \( \tau \) denotes an element of \( \text{Mng} \, \tau \). Actually, this is a rather conventional notion. In the simpler setting, complete specification of the semantics of a language requires not only a set \( S \) of stores, but also a function \( A \in \text{Id} \rightarrow \tau \), where \( \text{Id} \) is the set of identifiers. The function \( A \) has two uses. First, it is used to assign types to free occurrences of identifiers, enabling the type of the entire phrase to be determined. Second, \( A \) determines a predomain \( E \) of environments by taking \( E \) to be the product in \( \text{Pdom} \) of the \( \text{Id} \)-indexed collection \( \{ \text{Mng} \, (A \, x) \mid x \in \text{Id} \} \). Thus, each element of \( E \) associates a meaning of the right type to each identifier. Then even in this simpler setting we conceive of the denotation of a phrase of type \( \tau \) as being a (continuous) function from \( E \) to \( \text{Mng} \, \tau \). We have thus been led to the idea that the semantics of a phrase is a morphism in a category from an environment object to a meaning object. We will give more details shortly, but at least we need not worry that \( \text{Mng} \, \tau \) is a functor rather than a set of elements.

The idea that each identifier possesses a unique phrase type is workable in some contexts, but it is always unpalatable because it robs the programmer of the ability to bind any identifier to any phrase type. Furthermore, it is not in the spirit of ALGOL-like languages since an identifier should have
the ability to be bound to different phrase types in different blocks. The assignment of types to identifiers should be dynamic. Before program execution begins, no identifier has a type; during execution some have types, and some don't. This leads us to introduce $A$, the set of phrase type assignments. A phrase type assignment $\alpha$ is a function

$$\alpha \in I \rightarrow T$$

where $I = \text{dom} \alpha$ is a finite set of identifiers. Thus,

$\text{emp}_T \in \phi + T$,

the phrase type assignment with empty domain, is the phrase type assignment most appropriate to the beginning of program execution.

Now let $L$ be a desugared version of an ALGOL-like language with block structure and higher-order procedures. Without getting into the exact algebraic nature of $L$, we wish to be a little more precise about what phases in $L$ denote. Keep in mind that for each phrase type $\tau \in T$, Mng $\tau$ is an object of the Cartesian closed category $\Sigma \Rightarrow P\text{dom}$, where $\Sigma$ is the category of state shapes. To avoid complicated subscripts, let $K = \Sigma \Rightarrow P\text{dom}$. Actually in Chapters VII and VIII we use a slightly different $K$, but this one is good enough for the time being. We will use $\text{Type}$ to stand for the function that assigns phrase types to phrases. However, since a phrase may contain free occurrences of identifiers, a type can be assigned to $l \in L$ only in the context of a phrase type assignment $\alpha \in A$. Hence, we see that
Type \( \varepsilon L \to (A \to T) \).

(Thus, it seems that we try occasionnally to assign a type to a phrase \( \ell \in L \) in a context \( \alpha \in A \) that does not give types to all the free identifiers in \( L \). This situation can be handled by introducing a nonsense type \( \text{ns} \in T \).)

Each phrase type assignment \( \alpha \) determines an environment \( \text{Env} \alpha \), which is a functor in \( \Sigma \to \text{Pdom} \), defined by taking the product in \( K \) of the \( (\text{dom} \alpha) \)-indexed collection of functors \( \{ \text{Mng}(\alpha x) \mid x \in \text{dom} \alpha \} \). If \( X \) is a store shape, then \( \text{Env}_\alpha X \) is the product in \( \text{Pdom} \) of the \( (\text{dom} \alpha) \)-indexed collection of predomains \( \{ \text{Mng}(\alpha x) X \mid x \in \text{dom} \alpha \} \), which is a "conventional environment."

Let us use \( \text{Semf} \) to stand for the function which assigns denotations to phrases. A denotation can only be determined in the context of a phrase type assignment and the result will be a morphism (natural transformation) in the functor category \( K \). Hence, we want

1) \( \text{Semf} \in L \to (A \to \text{Ar} K) \)

and

2) for all \( \ell \in L, \alpha \in A, \)

\[ \text{Semf} \ell \alpha \in \text{Env} \alpha \xrightarrow{K} \text{Mng} (\text{Type} \ell \alpha). \]

In particular, the denotation of a phrase \( \ell \in L \) in the context of a phrase type assignment \( \alpha \in L \) and a store shape \( X \in \text{Ob} \Sigma \) is a function from a "conventional environment" to a "conventional collection of meanings," i.e.

\[ \text{Semf} \ell \alpha X \in \text{Env} \alpha X \xrightarrow{\text{Pdom}} \text{Mng} (\text{Type} \ell \alpha) X. \]
Our final topic in this section is a discussion of how this approach lends itself to an explanation of the interactions between procedures and block structure. However, we must preface this with the definition of the functor

$$\text{hom}^Σ ∈ Σ^{\text{op}} → K$$

The usual hom functor is

$$\text{hom}_Σ ∈ Σ^{\text{op}} × Σ → \text{Set}. $$

Let

$$E ∈ \text{Set} → P\text{dom}$$

be the embedding functor which gives a discrete partial order to each set. By composing, we get

$$E ∘ \text{hom}_Σ ∈ Σ^{\text{op}} × Σ → P\text{dom}. $$

Then curry to get

$$\text{hom}^Σ ∈ Σ^{\text{op}} → (Σ → P\text{dom}).$$

Thus, for store shapes X and Y, $\text{hom}^Σ X Y$ is $\text{hom}_Σ(X,Y) = X \xrightarrow{Σ} Y$, equipped with a discrete partial order.

Let's take a look at a phrase $ℓ ∈ L$, which in the context of $α ∈ A$, has a procedural type; say

$$\text{Type } ℓ α = τ ⇒ θ$$

where $τ, θ ∈ T$. Recall that we have an equality of functors:

$$\text{Mng } (τ ⇒ θ) = \text{Mng } τ ⇒ \text{Mng } θ.$$ 

Therefore,

$$\text{Semf } ℓ α ∈ \text{Env } α \xrightarrow{K} \text{Mng } τ ⇒ \text{Mng } θ.$$
Suppose the procedure \( \ell \) is defined in a program in a block whose store shape is \( X \). In this context, the denotation of the procedure is

\[
\text{Semf } \ell \, \alpha \, X \in \text{Env } \alpha \, X \xrightarrow{P_{\text{dom}}} (\text{Mng } \tau \Rightarrow \text{Mng } \theta) \, X.
\]

A peek ahead at the proof that \( K \) is a Cartesian closed category shows that the predomain \((\text{Mng } \tau \Rightarrow \text{Mng } \theta) \, X\) is obtained by partially ordering the set of natural transformations

\[
(hom^\Sigma X) \times (\text{Mng } \tau) \xrightarrow{K} \text{Mng } \theta.
\]

Let \( e \in \text{Env } \alpha \, X \) be the conventional environment existing when \( \ell \) is defined. Then

\[
\text{Semf } \ell \, \alpha \, X \in e \in (hom^\Sigma X) \times (\text{Mng } \tau) \xrightarrow{K} \text{Mng } \theta.
\]

Now suppose the procedure \( \ell \) is called in an interior block whose store shape is \( Y \). Thus, there is an expansion

\[
\sigma \in X \xrightarrow{\Sigma} Y.
\]

The denotation of \( \ell \) constructed at the time of its definition should be clearly connected with an element of the predomain \( \text{Mng } \tau \, Y \Rightarrow \text{Mng } \theta \, Y \), which is the proper collection of conventional meanings at the time of the call. Note that

\[
\text{Semf } \ell \, \alpha \, X \in Y \in (X \xrightarrow{\Sigma} Y) \times (\text{Mng } \tau \, Y) \xrightarrow{P_{\text{dom}}} (\text{Mng } \theta \, Y).
\]

Using the fact that \( P_{\text{dom}} \) is a Cartesian closed category we obtain

\[
\text{Ab}_{P_{\text{dom}}} (\text{Semf } \ell \, \alpha \, X \in Y) \in (X \xrightarrow{\Sigma} Y) \xrightarrow{P_{\text{dom}}} (\text{Mng } \tau \, Y \Rightarrow \text{Mng } \theta \, Y).
\]

Finally, apply this function to the specific expansion.
connecting the shape \( X \) of the store at the time of definition to the shape \( Y \) of the store at the time of the call, and we obtain

\[
\text{Ab}_{\mathcal{P}_{\text{dom}}} (\text{Semf } \lambda x \in X \rightarrow Y) \circ \in \text{Mng} \Rightarrow \text{Mng} \Rightarrow Y,
\]
a conventional meaning for the procedure at the time of call.

We have captured here two important intuitions about the meanings of procedures:

1. The environment used to determine the meaning of a procedure is not the environment existing at the moment of the call, but is rather the environment existing when the procedure is defined. (An intuition described by other approaches as well.)

2. The meaning of a procedure is also dependent on the store existing at the moment of call, the store existing at the moment of definition, and the expansion connecting them.
CHAPTER II
TYPE ALGEBRAS

Our aim in this chapter is the creation of mathematical objects that define collections of phrase types assignable to meaningful program fragments. These objects are certain "type algebras" freely generated by posets whose elements are viewed as primitive phrase types. These free type algebras are adequate for the subsequent discussion, but they don't provide the final answer to the question "What is a type?" In particular we don't delve into the nature of recursively defined phrase types.

It might seem adequate to construct simply a set \( T \) of phrase types, and, in the light of Chapter I, a semantic description of a programming language would start by specifying an object \( \text{Mng} \tau \) of a category (e.g. \( \text{Set} \) or \( \Sigma \Rightarrow \text{Pdom} \)) for each \( \tau \in T \).

This approach has an important deficiency. We wish to study the role of implicit conversions, i.e. coercions, in semantics. To say there is an implicit conversion from phrase type \( \tau \) to phrase type \( \theta \) (a condition expressed succinctly and suggestively as \( \tau \leq \theta \)) means in part that any phrase of type \( \tau \) can meaningfully replace any occurrence of a phrase of type \( \theta \) in a program. This relation is surely reflexive and transitive, and it is difficult to think of
an instance in which it is not antisymmetric. Thus $T$ is not just a set, but rather is a poset. Furthermore, in giving the semantics of a language, one must give a conversion morphism from $\text{Mng } T$ to $\text{Mng } \emptyset$ whenever $T \subseteq \emptyset$. These conversion morphisms must mesh properly with the partial order on $T$. This meshing is best described by making the poset $T$ into a category (in the usual way that posets become categories) and then requiring that $\text{Mng}$ be a functor from $T$ to $\text{Set}$ or $\Sigma \Rightarrow \text{Pdom}$ or whatever category is used.

We begin by establishing some notation and discussing some construction for posets.

We use $\emptyset$ to denote both the empty set and the poset whose underlying set of objects is empty. Similarly we use $\{a\}$ to denote both the singleton set and the singleton poset whose sole object is $a$. We say that $P$ is a subposet of $Q$ (notation: $P \subseteq Q$) if $\text{Ob } P \subseteq \text{Ob } Q$ and $\text{Ar } P \subseteq \text{Ar } Q$; in this case the inclusion mapping is monotone and is consequently a functor. If the inclusion functor is full, i.e. for all $x, y \in \text{Ob } P$

$$x \leq_P y \iff x \leq_Q y,$$

then $P$ is a full subposet of $Q$ (notation: $P \subseteq Q$). The intersection $P \cap Q$ of posets $P$ and $Q$ is defined by $\text{Ob } (P \cap Q) = (\text{Ob } P) \cap (\text{Ob } Q)$ and $\text{Ar } (P \cap Q) = (\text{Ar } P) \cap (\text{Ar } Q)$. Thus
\[ x \leq_{P \times Q} y \text{ iff } x \leq_P y \text{ and } x \leq_Q y. \]

Of course, \( P \cap Q \) is a subposet of both \( P \) and \( Q \). If \( P \cap Q = \emptyset \), then the posets \( P \) and \( Q \) are disjoint.

Let \( <P,Q> \) be an ordered pair of posets. Form \( P \times_{\text{Cat}} Q \) and consider the ordered pair whose first component is the set of objects and whose second component is the set of arrows of \( P \times_{\text{Cat}} Q \). Alas, it is not in general a poset because

\[ \text{Ar} \ (P \times_{\text{Cat}} Q) \subseteq \{(\text{Ob } P) \times (\text{Ob } Q)\} \times ((\text{Ob } P) \times (\text{Ob } Q)) \]

which is not the same as being a subset of

\[ (\text{Ob } (P \times_{\text{Cat}} Q)) \times (\text{Ob } (P \times_{\text{Cat}} Q)) = ((\text{Ob } P) \times (\text{Ob } Q)) \times ((\text{Ob } P) \times (\text{Ob } Q)). \]

For posets we have an additional, and perhaps more natural, product \( P \times Q \) given by

1. \( \text{Ob } (P \times Q) = (\text{Ob } P) \times (\text{Ob } Q) \), and

2. \( (x, y) \leq_{P \times Q} (x', y') \text{ iff } x \leq_P x' \text{ and } y \leq_Q y' \).

The definition of \( \times \) has an obvious generalization to more than two factors. As categories, \( P \times_{\text{Cat}} Q \) and \( P \times Q \) are isomorphic with the isomorphism being

\[ I : P \times_{\text{Cat}} Q \to P \times Q, \]

the functor which is the identity function on objects, and which on arrows satisfies \( I (x \leq_P x', y \leq_Q y') = (x, y) \leq_{P \times Q} (x', y) \).

Given two disjoint posets, the utility of connecting them together by providing a common top, i.e., maximal element,
will soon be seen. So suppose $P$ and $Q$ are disjoint posets and $t \not\in (\text{Ob } P) \cup (\text{Ob } Q)$. The poset $P \ast t \ast Q$ is defined by

$$\text{Ob } (P \ast t \ast Q) = (\text{Ob } P) \cup \{t\} \cup (\text{Ob } Q)$$

and

$$x \leq_{p \ast t \ast q} y \text{ iff } x \leq_p y \text{ or } y = t \text{ or } x \leq_q y.$$ 

Observe that one way to completely specify a functor $F : P \ast t \ast Q \rightarrow C$ is to let $F(t) = \text{term}$ where term is a terminal object of $C$ and to define functors in $P \rightarrow C$ and $Q \rightarrow C$ that are the restrictions of $F$ to $P$ and $Q$.

The first proposition summarizes some evident properties of full subposets.

**Proposition 2.1:** Suppose $P \subseteq Q$ and $P' \subseteq Q'$.

1. $P^{\text{op}} \subseteq Q^{\text{op}}$.
2. $P \times P' \subseteq Q \times Q'$.
3. If $Q$ and $Q'$ are disjoint, and $t \not\in (\text{Ob } Q) \cup (\text{Ob } Q')$. then $Q$, $Q'$, and $P \ast t \ast P'$ are all full subposets of $Q \ast t \ast Q'$.

**Proof:** Trivial. □

A moment's thought shows that the union of two posets $P$ and $Q$ cannot be defined by mimicking the definition of their intersection because $(\text{Ar } P) \cup (\text{Ar } Q)$ is not in general a partial order on $(\text{Ob } P) \cup (\text{Ob } Q)$. However, this simple-minded approach is entirely adequate for dealing with unions of ascending chains of posets, a matter we now take up.

Consider an ascending chain

$$P_0 \subseteq P_1 \subseteq P_2 \subseteq \ldots$$
of posets, which gives rise to a diagram

\[
\begin{array}{c}
\text{J}_0 \\
P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \\
\end{array}
\]

where each \( \text{J}_i \) is an inclusion mapping. We can form the poset

\[
P = \cup \{ P_i \mid i \in \mathbb{N} \}
\]

by letting \( \text{Ob} \ P = \cup \{ \text{Ob} \ P_i \mid i \in \mathbb{N} \} \) and

\[
\text{Ar} \ P = \cup \{ \text{Ar} \ P_i \mid i \in \mathbb{N} \}.
\]

Thus

\[
x \leq_P y \iff x \leq_{P_i} y \text{ for some } i \in \mathbb{N}.
\]

Of course, \( P_i \subseteq P \) for each \( i \). Let \( K_i \in P_i \rightarrow P \)
be the inclusion mapping, so that

\[
\begin{array}{c}
P_i \\
\downarrow \text{J}_i \\
P_{i+1} \\
\end{array}
\]

\[
\begin{array}{c}
P_i \\
\downarrow \text{K}_i \\
P \\
\end{array}
\]

\[
\begin{array}{c}
P_i \\
\downarrow \text{K}_{i+1} \\
P \\
\end{array}
\]

commutes for all \( i \). Then \( \langle P, \{ K_i \} \rangle \) is a direct limit of
the diagram (*) above, i.e., for each category \( C \) and each
\( \mathbb{N} \)-indexed collection \( F_i \in P_i \rightarrow C \) of functors such that

\[
\begin{array}{c}
P_i \\
\downarrow \text{J}_i \\
P_{i+1} \\
\end{array}
\]

\[
\begin{array}{c}
P_i \\
\downarrow F_i \\
C \\
\end{array}
\]

\[
\begin{array}{c}
P_i \\
\downarrow F_{i+1} \\
C \\
\end{array}
\]
commutes for all $i$, there exists a unique functor $F : P \to C$ such that

\[ \begin{array}{c}
K_i \\
\downarrow \\
F_i \\
\downarrow \\
P_i \\
\downarrow \\
F \\
\downarrow \\
P
\end{array} \]

commutes for all $i$. Note that if each $P_i$ is a full subposet of $P_{i+1}$, then each $P_i$ is a full subposet of $P$. Also, if

\[ Q_0 \subseteq Q_1 \subseteq Q_2 \subseteq \ldots \]

is another ascending chain of posets and

\[ Q = \bigcup \{ Q_i \mid i \in \mathbb{N} \} \]

then

\[ P_0 \times Q_0 \subseteq P_1 \times Q_1 \subseteq P_2 \times Q_2 \subseteq \ldots \]

is an ascending chain and

\[ P \times Q = \bigcup \{ P_i \times Q_i \mid i \in \mathbb{N} \}. \]

We believe that the object $T$ containing the types for a language having coercion among its features should be a poset. If the language is to have a general procedure-definition facility, then for any phrase types $\tau$ and $\theta$ there should be a type $\tau \Rightarrow \theta$ for procedures which accept arguments (i.e. parameters) of type $\tau$ and produce results (i.e. have calls) of type $\theta$. How should $\Rightarrow$
interact with the partial order on \( T \)? A procedure of type \( \tau \Rightarrow \theta \) should also accept arguments of type \( \tau' \) if \( \tau' \leq \tau \); the result of such a procedure can be used in a context calling for a result of type \( \theta' \) if \( \theta \leq \theta' \). Thus if \( \tau' \leq \tau \) and \( \theta \leq \theta' \), then we want \( \tau \Rightarrow \theta \leq \tau' \Rightarrow \theta' \). So \( \Rightarrow \) should be antimonotone in its first argument and monotone in its second, or in category-theoretic language \( \Rightarrow \) is like a hom functor in that it is contravariant in its first argument and covariant in its second.

We have argued that \( \text{Mng} \) should be a functor:

\[
\text{Mng} \in T \rightarrow K,
\]

where \( K \) is a category with appropriate structure. One thing that \( K \) should have is a functor contravariant in its first argument and covariant in its second; we can call this functor \( \Rightarrow \), too. We want \( \text{Mng} \) to preserve \( \Rightarrow \), i.e.

\[
\text{Mng}(\tau \Rightarrow \theta) = \text{Mng} \tau \Rightarrow \text{Mng} \theta
\]

for all phrase types \( \tau \) and \( \theta \). To see how plausible this is, consider the case \( K = \text{Set} \) with \( \Rightarrow \) equal to the ordinary hom functor; then the set of meanings for phrases of type \( \tau \Rightarrow \theta \) should be the set of functions from \( \text{Mng} \tau \) to \( \text{Mng} \theta \).

In order to talk about type-checking we must be able to assign a nonsense type \( T_{ns} \) to parseable expressions which contain errors due to mismatched types. Since any expression can be used in a context for which nonsense
suffices, the nonsense type should be the top, i.e. the terminal object, of $T$. What should be the nature of $\text{Mng } T_{ns}$? Knowing that a phrase has a nonsense meaning amounts to asserting that its meaning contains no information. Thus $\text{Mng } T_{ns}$ should be a singleton set when $K = \text{Set}$, and it should be the analogue of a singleton set, i.e. a terminal object, for other possible $K$.

To make everything elegant, both $T$ and $K$ should be the same sort of objects, and $\text{Mng}$ should be a homomorphism from $T$ to $K$. This serves to motivate the following definitions.

An ordered triple \[ A = < |A|, A_{ns}, A_\rightarrow > \]
is a type algebra if

1. $|A|$ is a category, called the carrier of $A$,
2. $A_{ns}$ is a terminal object of $|A|$, called the nonsense object of the algebra, and
3. $A_\rightarrow \in |A|^{\text{op}} \times |A| \rightarrow |A|.$

Alternative notations for $A_\rightarrow <a,b>$ are $a \rightarrow b$ and, when $A$ is readily determined from context, $a \Rightarrow b$. Cartesian closed categories give the most accessible examples of type algebras. Thus, from a Cartesian closed category $K$ we obtain a type algebra $A$ by letting $|A| = K$, $A_{ns} = K_{\text{term}}$, and $A_\rightarrow = K_\rightarrow$.

A type algebra homomorphism consists of a domain $A$
and a codomain $B$, both of which are type algebras, and a functor $F$ (notation: $F \in A \xrightarrow{\text{Type}} \mathcal{A}_{\mathcal{L}} \rightarrow B$) such that

1. $F \in |A| \rightarrow |B|,$
2. $F \mathcal{A}_{ns} = B_{ns},$ and
3. the diagram

$$
\begin{array}{c}
|A| \xrightarrow{\mathcal{A}} |A|
\end{array}
\quad
\begin{array}{c}
|E| \xrightarrow{\mathcal{E}} |E|
\end{array}

\quad
\begin{array}{c}
F \xrightarrow{\mathcal{F}} F
\end{array}
$$

commutes.

For the purpose of assigning phrase types to program fragments we need a type algebra whose carrier is a poset. Generally we expect to start with a poset $P$ of primitive phrase types. For instance, we might take $P$ to be the poset described by the following Hasse diagram.

```
real-exp  
\downarrow  
real-var  |  int-exp  
\downarrow  |  \ 
int-var  |  \ 
\downarrow  
bool-exp  
\downarrow  
comm
```

Here $\text{exp, var, int, bool,}$ and $\text{comm}$ are abbreviations for expression, variable, integer, boolean, and command, respectively. A complete discussion of posets such as these
including motivation for the structure of the Hasse diagram above may be found in Reynolds [8]. However, the central ideas are that variables may be coerced into expressions, integers may be coerced into reals, boolean phrase types cannot be coerced into real or integer phrase types, and commands cannot be coerced into other phrase types. Of course, \( P \) is not a type algebra since it has no procedural types and no nonsense type. However, the following theorem asserts that each poset \( P \) generates a free type algebra \( T \) whose carrier is a poset. It is the free type algebra generated by the poset of primitive phrase types which provides the phrase types for program fragments.

A few comments on this theorem are in order. In the theorem \( Q \) is the poset of procedural phrase types. The theorem says that the set underlying the carrier of \( T \) is the disjoint union of the set of primitive phrase types, the singleton consisting of the nonsense type, and the procedural types. If \( \tau \) is a procedural phrase type, then there are uniquely determined phrase types \( \theta \) and \( \theta' \) such that \( \tau = \theta \Rightarrow \theta' \) because \( T_{\omega} \) is injective on objects. Therefore, if we ignore the partial order, \( T \) is a free (universal) algebra generated by \( \text{Ob}_P \) where the signature (i.e. ranked set of operators) consists of a single operator \( \Rightarrow \) of rank 0 and a single operator \( \Rightarrow \) of rank 2. In a different vein, notice how easy it will be to define the type algebra homomorphism
we need only give the functor which is the restriction of \( \text{Mng} \) to \( P \).

The proof of the theorem is an exercise in the exploitation of the properties of unions of ascending chains of posets.

**Theorem 2.2:** Let \( P \) be a poset. There is a type algebra \( T \) such that

1. \(|T|\) is a poset, \( T \to \in |T|^{op} \times |T| \to |T|\) is a full and faithful functor, \(|T| = P_s \cdot T_{ns} \cdot Q_s\), where \( Q_s \) is the image of \( T_{ns} \), and

2. \( T \) satisfies the following universal mapping property:

   let \( K \) be a type algebra, let \( \phi \in P \to |K| \), and let \( \text{incl} \in P \to |T| \) be the inclusion functor; then there exists a unique type algebra homomorphism

   \( F \in T \xrightarrow{\text{Type Alg}} K \) such that

\[
\begin{array}{ccc}
P & \xrightarrow{\phi} & |K| \\
\downarrow{\text{incl}} & & \\
|T| & \xrightarrow{F} & |
\end{array}
\]


Proof: Let \( t \) be such that \( t \not\in \text{Ob} P \), and let \( a \) be such that no element of \((\text{Ob} P) \cup \{t\}\) is an ordered triple \( <a, x, y> \) whose first component is \( a \).
We start by constructing inductively a sequence

\[ Q_0, T_0, Q_1, T_1, Q_2, T_2, \ldots \]

of posets such that, for all \( i \), \( P \cap Q_i = \phi \). Let

\[ Q_0 = \phi. \]

Assuming that the sequence is constructed through \( Q_i \), let

\[ T_i = P \times t \times Q_i. \]

Assuming that the sequence is constructed through \( T_i \), let

\[ Q_{i+1} = \{ a \} \times T_i^{\text{op}} \times T_i. \]

We claim that

\[ Q_0 \subseteq Q_1 \subseteq Q_2 \subseteq \ldots \]

and

\[ T_0 \subseteq T_1 \subseteq T_2 \subseteq \ldots. \]

Obviously \( Q_0 \subseteq Q_1 \). Suppose

\[ Q_0 \subseteq \ldots \subseteq Q_{i-1} \subseteq Q_i \]

and

\[ T_0 \subseteq \ldots \subseteq T_{i-1}. \]

It follows from Proposition 2.1 that

\[ T_{i-1} = P \times t \times Q_{i-1} \subseteq P \times t \times Q_i = T_i. \]

Suppose

\[ Q_0 \subseteq \ldots \subseteq Q_i \]

and
\[ T_0 \subseteq \ldots \subseteq T_i. \]

Use Proposition 2.1 again to see

\[ Q_i = \{a\} \times T_{i-1}^{\text{op}} \times T_i \subseteq \{a\} \times T_i^{\text{op}} \times T_i = Q_{i+1}. \]

This proves the claim.

Let \(|T| = \cup \{T_i \mid i \in \mathbb{N}\}\) and \(Q = \cup \{Q_i \mid i \in \mathbb{N}\}\).

Clearly \(Q = \{a\} \times |T|^{\text{op}} \times |T|\). By the choice of \(a\),
\(P \cap Q = \emptyset\), and, by the choice of \(t\), \(t \notin (\text{Ob } P) \cup (\text{Ob } Q)\).

It is now apparent that \(\text{Ob } |T|\) is a disjoint union:

\[ \text{Ob } |T| = (\text{Ob } P) \cup \{t\} \cup (\text{Ob } Q). \]

Also,

\[ \tau \leq_{|T|} \theta \iff \tau \leq_{T_i} \theta \text{ for some } i \]
\[ \iff \tau \leq_{P} \theta \text{ or } t = \theta \text{ or } \tau \leq_{Q_i} \theta \text{ for some } i \]
\[ \iff \tau \leq_{P} \theta \text{ or } t = \theta \text{ or } \tau \leq_{Q} \theta \]

Therefore, \(|T| = P \ast t \ast Q\).

For each \(i\), let \(H_i \in T_i^{\text{op}} \times T_i \rightarrow Q_{i+1}\) be the obvious

poset isomorphism given by

\[ H_i = \langle a, \tau, \theta \rangle. \]

Consider the commutative diagram

\[ \begin{array}{ccc}
T_0^{\text{op}} \times T_0 & \rightarrow & T_1^{\text{op}} \times T_1 & \rightarrow & T_2^{\text{op}} \times T_2 & \rightarrow & \ldots \\
H_0 & & H_1 & & H_2 & & \\
Q_0 & \rightarrow & Q_1 & \rightarrow & Q_2 & \rightarrow & \ldots
\end{array} \]
where the horizontal maps are inclusions. After noting that

\[ |T|^\text{op} \times |T| = \bigcup_i \{ T_i^\text{op} \times T_i \mid i \in \mathbb{N} \}, \]

we assert that the poset isomorphism \( H \in |T|^\text{op} \times |T| \to \mathcal{Q} \)
given by \( H(x, y) = \langle a, x, y \rangle \) is the only functor such that

\[
\begin{array}{ccc}
T_i^\text{op} \times T_i & \xrightarrow{\text{incl}} & |T|^\text{op} \times |T| \\
\downarrow H_i & & \downarrow H \\
\mathcal{Q}_{i+1} & \xrightarrow{\text{incl}} & \mathcal{Q}
\end{array}
\]

commutes for all \( i \). Since \( H \) is an isomorphism it is full
and faithful.

For \( \tau, \theta \in \text{Ob } |T| \), we want the equation

\[ T_{<\tau, \theta>} = \langle a, \tau, \theta \rangle \]

to hold; so define \( T_{<\cdot, \cdot>} \in |T|^\text{op} \times |T| \to |T| \) so that

\[
\begin{array}{ccc}
|T|^\text{op} \times |T| & \xrightarrow{T_{<\cdot, \cdot>}} & |T| \\
\downarrow \text{incl} & & \downarrow \text{incl} \\
|T|^\text{op} \times |T| & \xrightarrow{H} & \mathcal{Q}
\end{array}
\]

commutes, where \( I \) is the canonical isomorphism and incl
is the inclusion functor. Clearly, the image of \( T_{<\cdot, \cdot>} \) is \( \mathcal{Q} \),
and \( T_{<\cdot, \cdot>} \), being a composition of full, faithful functors,
is itself full and faithful.
Since $t$ is a terminal object of $|T|$, we may let $T_{ns} = t$. Therefore $|T| = P \ast T_{ns} \ast Q$. This proves the first part of the theorem.

For the second part, let $K$ be a type algebra, and let $\phi \in P \to |K|$. We will define a sequence of functors $G_0, F_0, G_1, F_1, G_2, F_2, \ldots$ such that

$$G_i \in Q_i \to |K| \text{ and } F_i = T_i + |K|.$$  

Since $Q_0 = \phi$, the functor $G_0$ is uniquely determined.

Suppose we have constructed all functors through $G_i$. Recall $T_i = P \ast t \ast Q_i$, and let $F_i$ be the unique functor satisfying

1. $F_i t = K_{ns}$,
2. the restriction of $F_i$ to $P$ is $\phi$, and
3. the restriction of $F_i$ to $Q_i$ is $G_i$.

Suppose we have constructed all functors through $F_i$. By $T_i \in T_i^{op} \times T_i + T_i^{op} \times T_i$ we mean the canonical isomorphism.

Let $G_{i+1}$ be such that
commutes. Thus for $\tau, \theta \in \text{Ob } T_1$,

$$G_{i+1}^{a, \tau, \theta} = K_{\leq F_i^{\text{op}}} \tau, F_1 \theta.$$  

Next, we claim that the diagrams in the sequence

\[
\begin{array}{cccccc}
\text{incl} & Q_0 & \rightarrow & Q_1 & \rightarrow & Q_2 & \rightarrow & \cdots \\
\text{incl} & T_0 & \rightarrow & T_1 & \rightarrow & T_2 & \rightarrow & \cdots \\
G_0 & \rightarrow & G_1 & \rightarrow & G_1 & \rightarrow & \cdots \\
\text{incl} & F_0 & \rightarrow & F_1 & \rightarrow & F_2 & \rightarrow & \cdots \\
|K| & \rightarrow & |K| & \rightarrow & |K| & \rightarrow & \cdots \\
\end{array}
\]

are all commutative. The first diagram commutes because $Q_0 = \emptyset$. Suppose all diagrams commute through

\[
\begin{array}{ccc}
\text{incl} & Q_{i-1} & \rightarrow & Q_i \\
G_{i-1} & \rightarrow & G_i \\
|K| & \\
\end{array}
\]

then the commutativity of

\[
\begin{array}{ccc}
\text{incl} & T_{i-1} & \rightarrow & T_i \\
F_{i-1} & \rightarrow & F_i \\
|K| & \\
\end{array}
\]

(\textbf{**})

is immediate from the definitions of $F_{i-1}$ and $F_i$. Suppose all the diagrams through (**) commute; then from the commutativity of all the inner diagrams in
we get commutativity of the outer triangle.

By applying the earlier remarks on direct limits of ascending chains of posets, we see there exist unique functors $G$ and $F$ such that for all $i$ the diagrams

\[
\begin{array}{ccc}
\text{incl} & Q_i & \rightarrow & Q_i+1 \\
\downarrow & G & \downarrow & \text{incl} \\
Q_i & \rightarrow & |K|
\end{array}
\quad \begin{array}{ccc}
incl & |T| & \rightarrow & |T|+1 \\
\downarrow & F & \downarrow & \text{incl} \\
|T| & \rightarrow & |K|
\end{array}
\]

commute. The functors $G$ and $F$ are related by two commutative
diagrams. To obtain the first diagram, note that for each $i > 0$ the diagram

\[
\begin{array}{c}
\overset{\text{incl}}{Q_i \rightarrow T_{i-1}^{\text{op}} \times T_{i-1}} \rightarrow \overset{\text{incl}}{T_{i-1}^{\text{op}} \times T_{i-1}} \rightarrow \overset{\text{incl}}{T_{i-1}^{\text{op}} \times T_{i}} \rightarrow \overset{\text{incl}}{|T|^{\text{op}} \times |T|} \\
\overset{H^{-1}}{Q_i \rightarrow H^{-1}} \rightarrow \overset{I^{-1}}{|T|^{\text{op}} \times |T|} \rightarrow \overset{F^{\text{op}} \times F}{|T|^{\text{op}} \times |T|} \rightarrow \overset{G_i}{|K| \times |K|} \\
\end{array}
\]

commutes, and

\[
\phi = \overset{\text{incl}}{Q_0} \rightarrow Q \quad \overset{K_{\rightarrow} \circ (F^{\text{op}} \times F) \circ I^{-1} \circ H^{-1}}{K_{\rightarrow}} \rightarrow |K|
\]

is trivially commutative. Therefore the composite vertical arrow must equal $G$, i.e.
commutes. Also, for each $i$ the diagram

\[
\begin{array}{ccc}
Q_i & \xrightarrow{\text{incl}} & T_i \\
\downarrow & & \downarrow \\
Q & \xrightarrow{\text{incl}} & T \\
\end{array}
\]

commutes. Again, the composite vertical arrow must equal $G$, and we get a commutative diagram

\[
\begin{array}{ccc}
Q & \xrightarrow{\text{incl}} & |T| \\
\downarrow & & \downarrow \\
G & \xrightarrow{F} & F \\
\end{array}
\]

To conclude that $F$ is a type algebra homomorphism, there are three conditions that must be checked. First, from the definition of $F$ it is clear that $F \in |T| \to |K|$. Second,
since $F$ extends $F_0 \in T_0 + |K|$ we easily compute

$$F \downarrow_{ns} = F_0 \uparrow t = K_{ns}.$$

Finally, we must verify that the outer square is commutative in the diagram

and this follows because all the inner diagrams commute. Thus

$$F \in T \xrightarrow{\text{Type Alg}} K.$$

Notice that

commutes because $F$ extends $F_0$ which in turn extends $\phi$.

Finally we must verify the uniqueness of $F$. Let

$$F' \in T \xrightarrow{\text{Type Alg}} K$$
be an arbitrary type algebra homomorphism whose restriction to \( P \) is \( \phi \), and let
\[
G'_0, F'_0, G'_1, F'_1, G'_2, F'_2, \ldots
\]
be the restrictions of \( F' \) to
\[
Q'_0, T'_0, Q'_1, T'_1, Q'_2, T'_2, \ldots
\]
respectively. We will show by induction that
\[
G'_0 = G_0, F'_0 = F_0, G'_1 = G_1, F'_1 = F_1, G'_2 = G_2, \ldots
\]
Since \( Q'_0 = \emptyset, G'_0 = G_0 \). Suppose all the equalities through \( G'_i = G_i \) are true. We make three observations.

1. \( F'_i T'_n s = F'_i T'_n s = K'_n s \) since \( F' \) is a homomorphism.
2. The restriction of \( F'_i \) to \( P \) is the restriction of \( F' \) to \( P \), which is \( \emptyset \).
3. The restriction of \( F'_i \) to \( Q'_i \) is the restriction of \( F' \) to \( Q'_i \), which is \( G'_i = G_i \).

By the uniqueness of the construction of \( F'_i \), we get \( F'_i = F_i \).

Now suppose all the equalities through \( F'_i = F_i \) are true. Observe that in the following diagram all inner diagrams commute.
The reasons for commutativity of various diagrams is as follows:

(1) definition of $H$,
(2) $G_{i+1}'$ is the restriction of $F'$ to $Q_{i+1}$,
(3) $F'$ is a type algebra homomorphism,
(4) $F_i'$ is the restriction of $F'$ to $T_i$, and
(5) definition of $T_\Rightarrow$.

Thus the outer square commutes. By the definition of $G_{i+1}'$, we see that $G_{i+1}' = G_{i+1}$. Therefore all the equalities in the sequence are valid. Since $|T| = \bigcup \{ T_i | i \in \mathbb{N} \}$ and the restriction of $F'$ to $T_i$ in all cases equals the restriction of $F$ to $T_i$, we conclude $F' = F$. $\square$
CHAPTER III
SYNTAX, SUBSTITUTION, AND TYPE-CHECKING

In this chapter we will attempt to deal with the essentially syntactic problems of language definition, substitution, and type-checking.

First we must properly define the class of languages that are the subject of this work. On an intuitive level, a language consists of all combinations of symbols which can be parsed. Just as a natural language has many nonsensical or ill-defined sentences, so also a programming language should contain nonexecutable or meaningless expressions. This enables us to illuminate the nature of meaningful expressions by comparing them with similar nonsensical ones. Thus we will be able to address in general terms the question of why the assignment command

$x := 3$

is meaningful in some contexts, whereas

$3 := x$

is not. It is difficult to even pose this question if $3 := x$ is not an expression of the language. Of course, a desirable feature of an implementation of a language is a mechanism, generally a type-checking facility, which identifies nonsensical programs at compile time. The final portion of this chapter is devoted to an exposition of the view that type-checking is nothing more than the application of a
simply defined homomorphism to a phrase.

For a programming language we consider an expression to be parsable if its applicative structure can be determined. Thus, we are led to study typed $\lambda$-calculi. However, the languages are nonstandard in the sense that in each there is a single set of identifiers rather than a set of identifiers for each phrase type. This serves to intensify their similarity to implemented languages, particularly those of the ALGOL family, which permit different occurrences of an identifier in a program to have different types. Each language is syntactically determined by three parameters:

1. $P$, the poset of primitive phrase types,
2. $\text{Id}$, an infinite set of identifiers, and
3. $G$, a set called the generating set of the language, whose elements are termed linguistic constants.

These three parameters suffice for language definition, but other parameters must be specified in order to deal with substitution and type-checking in this chapter and with semantics in the next. The resulting language is called a coercive typed $\lambda$-calculus.

In the remainder of this chapter we assume that the poset $P$, the infinite set $\text{Id}$, and the set $G$ are all fixed. We now turn to the details of how a language $L$ arises from $P$, $\text{Id}$, and $G$.

As shown in the previous chapter, the poset $P$ of primitive phrase types generates a free type algebra
\( T = \langle |T|, T_{ns}, T_\tau \rangle \). We will use \( \text{Ob } |T| \) as the collection of all phrase types for \( L \). For ease of reading, we usually write \( \tau \in T \) rather than \( \tau \in \text{Ob } |T| \), \( \tau \leq \theta \) rather than \( \tau \leq |T| \theta \), \( ns \) rather than \( T_{ns} \), and \( \tau = \theta \) rather than \( T_{\tau, \theta} \).

At this point we should say a word about our notation for one-sorted algebras. A signature \( \Sigma \) is just a ranked set; thus each \( \sigma \in \Sigma \) is assigned a rank or arity which is non-negative integer. The elements of \( \Sigma \) are called operators, and the set of operators of arity \( n \) is denoted \( \Sigma_n \). In this setting, a \( \Sigma \)-algebra

\[
A = \langle |A|, \{ A_\sigma \mid \sigma \in \Sigma \} \rangle
\]

is formally a pair consisting of a set \( |A| \), the carrier of the algebra, and a collection \( \{ A_\sigma \mid \sigma \in \Sigma \} \) of interpretations of the operators in \( \Sigma \) such that

\[
A_\sigma \in |A|^n \rightarrow |A|,
\]

if \( \sigma \in \Sigma_n \). We usually write \( x \in A \) rather than \( x \in |A| \).

Homomorphisms of \( \Sigma \)-algebras are defined in the obvious way. For details, see Cohn [2].

From \( T \) and \( \text{Id} \) we get a signature \( \Lambda \) for ordinary one-sorted algebras. The collection of 0-ary operators of \( \Lambda \) is \( \text{Id} \). For each \( \tau \in T \) and \( x \in \text{Id} \) there is a single unary operator denoted \( \lambda x : \tau \). Finally there is a single binary operator denoted \( ap \). Thus

\[
\Lambda_0 = \text{Id}, \quad \Lambda_1 = \{ \lambda x : \tau \mid x \in \text{Id}, \tau \in T \}, \quad \Lambda_2 = \{ ap \}.
\]

The role of the 0-ary operators is to insure that identifiers
are expressions. The unary operator \( \lambda x: \tau \) creates procedures (\( \lambda \)-expressions) designed to be meaningfully applied to arguments which can be coerced into type \( \tau \). The operator \( \text{ap} \) enables us to freely apply any expression to any other.

Let
\[
L = \langle |L|, \{ L_\omega \mid \omega \in \Lambda \rangle
\]
be a free \( \Lambda \)-algebra generated by \( G \), the generating set of linguistic constants. For example, suppose \( \text{int-exp} \) is a primitive phrase type, \( x \in \text{Id} \), and \( 0, 1, + \) are all in \( G \). An expression of \( L \) is
\[
L_{\text{ap}} < L_{\lambda x: \text{int-exp}} (L_{\text{ap}} < L_{\text{ap}} <+1>, L_x) >, 0 >,
\]
which is a desugared version of
\[
(\lambda x: \text{int-exp} \ (1+x)) \ 0 .
\]
(We are viewing + as a curried operator that applies to an integer and returns a function from integers to integers.)

In the interest of clarity we shall use a conventional \( \lambda \)-calculus notation for elements of \( L \), and we will call attention to occurrences of this conventional notation with emphatic brackets. Thus, if \( x \in \text{Id} \), \( \tau \in T \), \( l \in L \) and \( m \in L \), then
\[
[x] \text{ stands for } L_x ,
\]
\[
[\lambda x: \tau . l] \text{ stands for } L_{\lambda x: \tau} l , \text{ and}
\]
\[
[lm] \text{ stands for } L_{\text{ap}} <l, m> .
\]

An important notion in any treatment of the \( \lambda \)-calculus is the set of identifiers that occur free in an expression.
The first use we make of the algebraic structure of $L$ is to cast the assignment of sets of free identifiers to expressions as a $\lambda$-algebra homomorphism. Define the $\lambda$-algebra $F$ by letting

$$|F| = \text{set of finite subsets of Id},$$

and defining

$$F_x = \{x\}, \text{ where } x \in \text{Id},$$

$$F_{\alpha} < F, H > = F \cup H, \text{ where } F, H \in F, \text{ and }$$

$$F_{\lambda x : \tau} F = F - \{x\}, \text{ where } x \in \text{Id}, \tau \in \tau, F \in F.$$

There are no free occurrences of any identifiers in any linguistic constant. This leads us to define the $\Lambda$-algebra homomorphism

$$\begin{array}{c}
\text{Free } : L \\ \Lambda - \text{Alg}
\end{array} \rightarrow F$$

to be the unique homomorphism such that, for all $g \in G$,

$$\text{Free } g = \emptyset.$$

The following theorem then shows that Free behaves exactly as it should.

**Theorem 3.1**: Free is the unique function such that for all $g \in G$, $x \in \text{Id}$, $\tau \in \tau$, $\ell \in L$, $m \in L$ the following four equations hold:

(i) $\text{Free } g = \emptyset$, 

(ii) $\text{Free } [x] = \{x\}$, 

(iii) $\text{Free } [\ell, m] = (\text{Free } \ell) \cup (\text{Free } m)$, 

(iv) $\text{Free } [\lambda x : \tau. \ell] = (\text{Free } \ell) - \{x\}$. 
Proof: According to the preceding description of Free, (i) holds by definition and (ii)-(iv) follow because Free is a homomorphism. Conversely, any function satisfying (ii)-(iv) is a homomorphism, and Free is the only homomorphism satisfying (i). \[ \square \]

The next topic to be explored is the effect on the treatment of substitution caused by the view that a language is a free $A$-algebra. Basically substitution involves the creation of a new phrase from an assignment of phrases to identifiers. It is complicated by the fact that bound occurrences of identifiers are insulated from the effects of the substitution unless a name change is necessitated to avoid binding an occurrence which should be free.

A phrase assignment is a function $f \in F \rightarrow |L|$ whose domain is a finite set of identifiers, i.e., $F \in F$. The collection of all phrase assignments is denoted $|L|^*$. A particularly noteworthy phrase assignment is the empty phrase assignment whose domain is the empty set; it is denoted by

$$\text{emp}_L \in \emptyset \rightarrow |L|$$

Use of the empty phrase assignment to determine a function from $|L|$ to $|L|$ by substitution should yield the identity function.

A function $f \in \text{Id} \rightarrow |L|$, whose domain is the set of all identifiers, is called a comprehensive phrase assignment. The set of all comprehensive phrase assignments is denoted $|L|^\circ$. Each $f \in |L|^*$ determines a canonical $f^\circ \in |L|^\circ$ via the equation
\[
\begin{align*}
    f^\infty x &= \begin{cases} 
        fx & \text{if } x \in \text{dom } f \\
        [x] & \text{if } x \notin \text{dom } f.
    \end{cases}
\end{align*}
\]

Thus, \(\text{emp}_L^\infty\) resembles the identity function because, for all \(x \in \text{Id}\),

\[
    \text{emp}_L^\infty x = [x] .
\]

In order to describe the condition which determines when a substitution involves a change of bound identifiers, we introduce

\[
    \text{clashset} \; \varepsilon \; |L|^* \times \text{Id} \times |L| + |F|
\]

which is defined by

\[
    \text{clashset} \; <f, x, \ell> = \cup \{\text{Free } (f^\infty y) \mid y \in \text{Free } \ell \text{ and } y \neq x\}
\]

The idea is this: our intuition tells us that the substitution associated with a phrase assignment \(f\) will involve changing the bound identifier \(x\) in \([\lambda x : \tau. \ell]\) if \(x \in \text{clashset} \; <f, x, \ell>\). Note that

\[
    \text{clashset} \; <f, x, \ell> = \cup \{\text{Free } (f^\infty y) \mid y \in \text{Free}[\lambda x : \tau. \ell]\}
\]

In the remainder of this chapter, and in the next chapter, we assume that the set \(\text{Id}\) of identifiers comes equipped with a function

\[
    \text{newid} \; \varepsilon \; |F| + \text{Id}
\]

such that, for any finite set \(F\) of identifiers,

\[
    \text{newid } F \notin F .
\]

The function \(\text{newid}\) provides a mechanism for changing bound identifiers when required.
Our goal is to define and illuminate a function

\[
\text{Sub} \in |L|^* \to |L| \to |L|
\]

that accepts a phrase assignment and produces a substitution function which maps phrases to phrases. Certainly, three properties we expect of substitution are that it does not change constants, it replaces phrases consisting of a single identifier with an appropriate expression, and it acts on an application by acting on the operator and the operand separately. The action of a substitution function on \(\lambda\)-expressions is more delicate. As is well known, sometimes renaming of bound identifiers is necessary to avoid identifier collisions. For instance, if a phrase assignment maps \(y\) to a phrase which has \(x\) among its free identifiers, then the result of applying the associated substitution function to \([\lambda x : \tau. \ [y]]\) should involve changing \(x\) to another identifier. Furthermore, in our approach to substitution we opt never to do unnecessary renaming. Thus, we expect Sub to be found among the members of the set of functions \(S \in |L|^* \to |L| \to |L|\) such that, for all \(f \in |L|^*\), the following conditions hold:

(Sub1) For all \(g \in G\), \(S \circ g = g\).

(Sub2) For all \(x \in \text{Id}\), \(S \circ [x] = f^x\).

(Sub3) For all \(\ell, m \in L\)

\[
S \circ [\ell m] = [[(S \circ \ell) (S \circ m)].
\]
(Sub4) For all \( x \in \text{Id}, \tau \in T, \ell \in L \),

\[
S f \llbracket \lambda x : \tau . \ell \rrbracket = \begin{cases} 
\llbracket \lambda x : \tau . \left( S \left[ f \mid x : [x] \right] \ell \right) \rrbracket & \text{if } x \notin \text{clashset} \langle f, x, \ell \rangle \\
\llbracket \lambda z : \tau . \left( S \left[ f \mid x : [z] \right] \ell \right) \rrbracket & \text{otherwise, where} \\
\quad \quad \quad z = \text{newid} (\text{clashset} \langle f, x, \ell \rangle).
\end{cases}
\]

We have a general method for defining \( \lambda \)-algebra homomorphisms whose domain is \( L \). Do we expect \( \text{Sub} f \) to be a \( \lambda \)-algebra homomorphism from \( L \) to \( L \) for each \( f \in |L|^* \)? The answer is no, because an affirmative answer would imply that for \( x \in \text{Id}, \tau \in T, \ell \in L \),

\[
\text{Sub} f \llbracket \lambda x : \tau . \ell \rrbracket = \llbracket \lambda x : \tau . (\text{Sub } f \ell) \rrbracket,
\]

an equation at odds with (Sub4). In fact, the complexity of condition (Sub4) makes it impossible to simply fix \( f \in |L|^* \) and then define \( \text{Sub} f \) by induction on the structure of its argument because \( \text{Sub } f \llbracket \lambda x : \tau . \ell \rrbracket \) is not expressed in terms of \( \text{Sub } f \ell \).

The class of functions satisfying the above four conditions is not too bizarre, as the next proposition shows.

**Proposition 3.2:** Suppose \( S \in |L|^* \rightarrow |L| \rightarrow |L| \) satisfies (Sub1), (Sub2), (Sub3), and (Sub4). Then \( S \text{ emp}_L \) is the identity function on \( |L| \).

**Proof:** By induction on the structure of \( \ell \in L \) we will show that, for all \( f \in |L|^* \) such that \( f^\infty = \text{emp}_L^\infty \), \( S f \ell = \ell \).
From this the theorem follows.

(1) Let \( \ell = g \) where \( g \in G \). Then \( S f g = g \) by (Sub1).

(2) Let \( \ell = [x] \) where \( x \in \text{Id} \). If \( f^\infty = \text{emp}^\infty_L \), then
\[
S f [x] = f^\infty x = \text{emp}^\infty_L x = [x]
\]
by (Sub2).

(3) Let \( \ell = [mn] \) where \( m, n \in L \). If \( f^\infty = \text{emp}^\infty_L \),
then
\[
S f [mn] = [(S f m) (S f n)]
\]
= \([mn]\), by the induction hypothesis.

(4) Let \( \ell = [\lambda x : \tau . \! m] \) where \( x \in \text{Id} \), \( \tau \in \cal T \), \( m \in L \). Suppose
\( f^\infty = \text{emp}^\infty_L \). Then \( x \notin \text{clashset} <f, x, m> \) because
\[
\text{clashset} <f, x, m>
= \cup \{\text{Free} \ (f^\infty y) \mid y \in \text{Free} m, y \neq x\}
= \cup \{\text{Free} \ [y] \mid y \in \text{Free} m, y \neq x\}
= \{y \mid y \in \text{Free} m, y \neq x\}.
\]

Therefore
\[
S f [\lambda x : \tau . \! m] = [\lambda x : \tau . \ (S f \ x : [x]) \ m]
= [\lambda x : \tau . \! m],
\]
because of the induction hypothesis and
\[
[f \ x : [x]]^\infty = f^\infty = \text{emp}^\infty_L.
\]

It should come as no surprise that there is a \( \Lambda \)-algebra homomorphism connected with substitution. Its codomain,
the \( \Lambda \)-algebra \( H \), is suggested by observing that there is a one-to-one correspondence between \( \|L\|^* \to \|L\| \to \|L\| \) and
$|L| + |L|^* + |L|$. Thus, we take the carrier of $H$ to be

$$|H| = |L|^* + |L|,$$

and we describe the interpretations of the operators by the equations

$$H_x f = f^x \cdot x \text{ where } x \in \text{Id}, f \in |L|^*,$$

$$H_{ap} <h, j> f = [(hf)(jf)] \text{ where } h, j \in H, f \in |L|^*,$$

$$H_{\lambda x : \tau} h f = \begin{cases} 
[\lambda x : \tau. (h [f | x : [x]])], & \text{if } x \notin \text{clashset} <f, x, h \text{ emp}_L> \\
[\lambda z : \tau. (h [f | x : [z]])], & \text{otherwise,}
\end{cases}$$

where $x \in \text{Id}$, $\tau \in \text{T}$, $h \in H$, $f \in |L|^*$, and $z = \text{newid} (\text{clashset} <f, x, h \text{ emp}_L>)$.

Let

$$\text{Presub } \in L \xrightarrow{\Lambda - \text{Alg}} H$$

be the unique $\Lambda$-algebra homomorphism such that for all $g \in G$, Presub $g$ is the constant function:

$$\text{Presub } g f = g \text{ for all } f \in |L|^*.$$

Now define

$$\text{Sub } \in |L|^* + |L| + |L|$$

by

$$\text{Sub } f \ell = \text{Presub } \ell f$$

for all $f \in |L|^*$, $\ell \in \ell L$. 

The next theorem asserts that the substitution function determined by a phrase assignment \( f \) depends only on the canonical extension of \( f \) to a comprehensive phrase assignment.

**Theorem 3.3:** For all phrase assignments \( f \) and \( j \), \( f^\infty = j^\infty \) implies \( \text{Sub } f = \text{Sub } j \).

**Proof:** The theorem will follow if we can prove by induction on the structure of \( \ell \in \mathcal{L} \) that for all phrase assignments \( f \) and \( j \), \( f^\infty = j^\infty \) implies \( \text{Sub } f \ell = \text{Sub } j \ell \). The fact that both \( f \) and \( j \) are universally quantified in the induction hypothesis is crucial to the proof.

1. Let \( \ell = g \) where \( g \in G \). Then, for all \( f, j \),
   \[
   \text{Sub } f g = \text{Presub } g \ f = g = \text{Presub } g \ j = \text{Sub } j \ g.
   \]

2. Let \( \ell = [x] \) where \( x \in \text{Id} \). Then \( f^\infty = j^\infty \) implies
   \[
   \text{Sub } f \ [x] = \text{Presub } [x] \ f = \mathcal{H}_x f = f^\infty x = j^\infty x = \text{Sub } j \ [x].
   \]

3. Let \( \ell = [mn] \) where \( m, n \in \mathcal{L} \). Then \( f^\infty = j^\infty \) implies
   \[
   \text{Sub } f \ [mn] = \text{Presub } [mn] \ f
   = \mathcal{H}_{ap} \ <\text{Presub } m, \text{Presub } n> \ f
   = [[(\text{Presub } m \ f) \ (\text{Presub } n \ f)]
   = [(\text{Sub } f \ m) \ (\text{Sub } f \ n)]
   = [(\text{Sub } j \ m) \ (\text{Sub } j \ n)],
   \]
   by the induction hypothesis,
   \[
   = \text{Sub } j \ [mn].
   \]

4. Let \( \ell = [\lambda x: \tau. \ m] \) where \( x \in \text{Id} \), \( \tau \in \mathcal{T} \), \( m \in \mathcal{L} \).

Suppose \( f^\infty = j^\infty \). If \( x \notin \text{clashset } <f, x, \text{Presub } m \text{ emp}_L> \), then
Sub \( f [\lambda x : \tau. m] \) = Presub \( [\lambda x : \tau. m] f \)

\[ = H_{\lambda x : \tau}(\text{Presub } m) f \]
\[ = [\lambda x : \tau. (\text{Presub } m [f \mid x : [x]])] \]
\[ = [\lambda x : \tau. (\text{Sub } [f \mid x : [x]] m)] \]
\[ = [\lambda x : \tau. (\text{Sub } [j \mid x : [x]] m)] \]

because of the induction hypothesis and
\[ [f \mid x : [x]]^{\infty} = [j \mid x : [x]]^{\infty}, \]
\[ = [\lambda x : \tau. (\text{Presub } m [j \mid x : [x]])] \]
\[ = H_{\lambda x : \tau}(\text{Presub } m) j, \]

because clashset \(<j, x, \text{Presub } m \text{ emp}_L>\)
\[ = \text{clashset } <f, x, \text{Presub } m \text{ emp}_L>, \]
\[ = \text{Sub } j [\lambda x : \tau. m]. \]

If \( x \in \text{clashset } <f, x, \text{Presub } m \text{ emp}_L>, \) then a similar argument gives \( \text{Sub } f [\lambda x : \tau. m] = \text{Sub } j [\lambda x : \tau. m]. \)

Our next task is to prove that the substitution function induced by the empty phrase assignment is the identity function on \( |L| \). We base the proof on the obvious observation that the identity function is a homomorphism.

**Theorem 3.4:** Sub \( \text{emp}_L = 1_L \).

**Proof:** We claim Sub \( \text{emp}_L \) is a \( \Lambda \)-algebra homomorphism.

This is implied by the following three computations.

(1) Let \( x \in \text{Id}. \) Then

\[ \text{Sub } \text{emp}_L [x] = \text{Presub } [x] \text{ emp}_L \]
\[ = H_x \text{ emp}_L \]
\[ = \text{emp}_L^{\infty} x \]
\[ = [x]. \]
(2) Let \( \ell, m \in L \). Then
\[
\text{Sub } \mathsf{emp}_L \[ \ell m \] = \text{Presub } \[ \ell m \] \mathsf{emp}_L
\]
\[
= \mathcal{H}_{ap} <\text{Presub } \ell, \text{Presub } m> \mathsf{emp}_L
\]
\[
= \llbracket (\text{Presub } \ell \mathsf{emp}_L) (\text{Presub } m \mathsf{emp}_L) \rrbracket
\]
\[
= \llbracket (\text{Sub } \mathsf{emp}_L \ell) (\text{Sub } \mathsf{emp}_L m) \rrbracket.
\]

(3) Let \( x \in \text{Id}, \tau \in \mathcal{T}, \ell \in L \). Observe that
\[
x \notin \text{clashset } <\mathsf{emp}_L, x, \text{Presub } \ell \mathsf{emp}_L>.
\]
because
\[
\text{clashset } <\mathsf{emp}_L, x, \text{Presub } \ell \mathsf{emp}_L>
\]
\[
= \bigcup \{ \text{Free } (\mathsf{emp}_L) y \mid y \in \text{Free } (\text{Presub } \ell \mathsf{emp}_L), y \neq x \}
\]
\[
= \bigcup \{ \text{Free } y \mid y \in \text{Free } (\text{Presub } \ell \mathsf{emp}_L), y \neq x \}
\]
\[
= \{ y \mid y \in \text{Free } (\text{Presub } \ell \mathsf{emp}_L), y \neq x \}.
\]
Thus
\[
\text{Sub } \mathsf{emp}_L \[ \lambda x : \tau. \ell \] = \text{Presub } \[ \lambda x : \tau. \ell \] \mathsf{emp}_L
\]
\[
= \mathcal{H}_{\lambda x : \tau} (\text{Presub } \ell) \mathsf{emp}_L
\]
\[
= \llbracket \lambda x : \tau. (\text{Presub } \ell \mathsf{emp}_L \mid x : [x]) \rrbracket
\]
\[
= \llbracket \lambda x : \tau. (\text{Sub } \mathsf{emp}_L \mid x : [x]) \ell \rrbracket
\]
\[
= \llbracket \lambda x : \tau. (\text{Sub } \mathsf{emp}_L \ell) \rrbracket, \text{ by Theorem 3.3.}
\]

So \( \text{Sub } \mathsf{emp}_L \) is a \( \Lambda \)-algebra homomorphism. By comparing the actions of \( \text{Sub } \mathsf{emp}_L \) and \( 1_L \) on generators
\( (\text{Sub } \mathsf{emp}_L \ g = \text{Presub } g \mathsf{emp}_L = g = 1_L \ g \text{ for } g \in G) \), we see
\( \text{Sub } \mathsf{emp}_L = 1_L \). 

Now we are ready to prove the most important theorem on substitution.

Theorem 3.5: \( \text{Sub} \) is the unique function in
\( |L|^* \to |L| \to |L| \) such that for all \( f \in |L|^* \), conditions (Sub1), (Sub2), (Sub3) and (Sub4) are all satisfied.
Proof: First we will verify that the four conditions are satisfied by Sub. In this proof let \( f \in \{L\}^* \), \( g \in G \), \( x \in \text{Id} \), \( \ell \in \{T\} \), \( m \in \{G\} \), and \( z = \text{newid} \) (clashset \( \langle f, x, \ell \rangle \)).

By definition of Presub on generators,

1. \( \text{Sub} f \, x = \text{Presub} \, g \, f \, g = g \).

Because Presub is a homomorphism,

2. \( \text{Sub} f \, [x] = \text{Presub} \, [x] \, f = \text{H}_x \, f = f^\infty \, x \),

3. \( \text{Sub} f \, [\ell m] = \text{Presub} \, [\ell m] \, f \)

   \[ = \text{H}_{\ell m} \, <\text{Presub} \, \ell, \text{Presub} \, m> \, f \]

   \[ = ([\text{Presub} \, \ell \, f] \, (\text{Presub} \, m \, f)) \]

   \[ = ([\text{Sub} \, f \, \ell] \, (\text{Sub} \, f \, m)), \text{ and} \]

4. if \( x \notin \text{clashset} \, \langle f, x, \ell \rangle \), then

   \( \text{Sub} f \, [\lambda x : \tau . \ell] = \text{Presub} \, [\lambda x : \tau . \ell] \, f \)

   \[ = \text{H}_{\lambda x : \tau} \, (\text{Presub} \, \ell) \, f \]

   \[ = ([\lambda x : \tau . (\text{Presub} \, \ell \, [f \mid x : [x]])]) \]

   because clashset \( \langle f, x, \text{Presub} \, \ell \, \text{emp}_\ell \rangle \)

   \[ = \text{clashset} \, \langle f, x, \ell \rangle \text{ by Theorem 3.4} \]

   \[ = ([\lambda x : \tau . (\text{Sub} \, [f \mid x : [x]] \, \ell])], \]

whereas, if \( x \in \text{clashset} \, \langle f, x, \ell \rangle \), then

\( \text{Sub} f \, [\lambda x : \tau . \ell] = \text{H}_{\lambda x : \tau} \, (\text{Presub} \, \ell) \, f \)

\[ = ([\lambda z : \tau . (\text{Presub} \, \ell \, [f \mid x : [z]])]) \]

(here again we apply Theorem 3.4)

\[ = ([\lambda z : \tau . (\text{Sub} \, [f \mid x : [z]] \, \ell])]). \]

So the four conditions are satisfied by Sub.

Let \( \text{Sub}' \) be any function in \( |L| \rightarrow |L| \rightarrow |L| \) satisfying the four conditions. Define \( \text{Presub}' \in |L| \rightarrow |L|^* \rightarrow |L| \).
by Presub' $\ell \ f = \text{Sub}' \ f \ell$. Using (Sub2) we see
\[ \text{Presub}' [x] \ f = \text{Sub}' f^\infty x = H_x f. \]

Thus
\[ \text{Presub}' [x] = H_x. \]

Using (Sub3) we see
\[ \text{Presub}' [\ell m] \ f = \text{Sub}' \ f [\ell m] \]
\[ = [(\text{Sub}' f \ell) (\text{Sub}' f m)] \]
\[ = [(\text{Presub}' \ell f) (\text{Presub}' m f)] \]
\[ = H_{\text{ap}} <\text{Presub}' \ell, \text{Presub}' m> f. \]

Thus
\[ \text{Presub}' [\ell m] = H_{\text{ap}} <\text{Presub}' \ell, \text{Presub}' m>. \]

Next we carry out some computations using (Sub4). There are two cases. First suppose $x \notin \text{clashset}<f, x, \ell>$; then
\[ \text{Presub}' [\lambda x: T. \ell] \ f = \text{Sub}' \ f [\lambda x: T. \ell] \]
\[ = [\lambda x: T. (\text{Sub}' [f | x::[x]] \ell)] \]
\[ = [\lambda x: T. (\text{Presub}' \ell [f | x::[x]] )] \]
\[ = H_{\lambda x: T} (\text{Presub}' \ell) f. \]

The last equality is valid because
\[ \text{clashset} <f, x, \text{Presub}' \ell \text{ emp}_L> = \text{clashset} <f, x, \text{Sub}' \text{ emp}_L \ell> \]
\[ = \text{clashset} <f, x, \ell>, \]
by Proposition 3.2.

Second, suppose $x \in \text{clashset} <f, x, \ell>$. Let
\[ z = \text{newid} (\text{clashset} <f, x, \ell>). \]

Then
\[ \text{Presub}' [\lambda x: T. \ell] \ f = \text{Sub}' \ f [\lambda x: T. \ell] \]
\[ = [\lambda z: T. (\text{Sub}' [f | x::[z]] \ell)] \]
\[ = [\lambda z: T. (\text{Presub}' \ell [f | x::[z]])] \]
\[ = H_{\lambda x: T} (\text{Presub}' \ell) f. \]
Thus

$$\text{Presub'} [\lambda x : \tau. \ell] = H_{\lambda x : \tau} (\text{Presub'} \ell)$$

Hence

$$\text{Presub'} \in L \xrightarrow{\Lambda - \mathcal{Alg}} H.$$ 

On generators the homomorphisms Presub' and Presub agree since for \(g \in G\), \(f \in L^*\),

$$\text{Presub'} g \circ f = \text{Sub'} f \circ g = g = \text{Presub} g \circ f.$$ 

Therefore \(\text{Presub'} = \text{Presub}\) and consequently \(\text{Sub'} = \text{Sub}\). □

The next theorem tells how to compute the set of identifiers occurring free in a phrase created by applying a substitution function to another phrase.

**Theorem 3.6:** Let \(\ell \in L\). For all \(f \in L^*\),

Free (Sub \(f \ell\)) = \(U \{\text{Free } (f^\infty y) \mid y \in \text{Free } \ell\}\).

**Proof:** We proceed by induction on \(\ell \in L\).

(1) Let \(\ell = g\) where \(g \in G\). For all \(f \in L^*\),

Free (Sub \(f \circ g\)) = Free \(g\), by (Sub1),

= \(\emptyset\), by Theorem 3.1,

= \(U \{\text{Free } (f^\infty y) \mid y \in \text{Free } g\}\).

(2) Let \(\ell = [x]\) where \(x \in \text{Id}\). For all \(f \in L^*\),

Free (Sub \(f [x]\)) = Free \(f_\infty x\), by (Sub2),

= \(U \{\text{Free } (f^\infty y) \mid y = x\}\)

= \(U \{\text{Free } (f^\infty y) \mid y \in \text{Free } [x]\}\),

by Theorem 3.1.

(3) Let \(\ell = [mn]\) where \(m, n \in L\). For all \(f \in L^*\),

Free (Sub \(f [mn]\))

= Free \((\text{Sub } f \circ m) \circ (\text{Sub } f \circ n)\), by (Sub3)
= \text{Free} (\text{Sub} f m) \cup \text{Free} (\text{Sub} f n),
    
    \text{by Theorem 3.1,}
    
= \bigcup \{\text{Free} (f^\infty y) \mid y \in \text{Free} m\} \cup \{\text{Free} n\},
    
    \text{by the induction hypothesis,}
    
= \bigcup \{\text{Free} (f^\infty y) \mid y \in \text{Free} \{mn\}\},
    
    \text{by Theorem 3.1.}

(4) Let \(L = [\lambda x: \tau. m]\) where \(x \in \text{Id}, \tau \in \mathcal{T}, m \in L\). Suppose 
    \(f \in \lambda L^*\) and \(z = \text{newid} (\text{clashset} <f, x, m>)\). If 
    \(x \notin \text{clashset} <f, x, m>\), then 

\[
\text{Free} (\text{Sub} f [\lambda x: \tau. m])
\]

\[
= \text{Free} [\lambda x: \tau. (\text{Sub} [f \mid x::[x]] m)], \text{by (Sub4),}
\]

\[
= \text{Free} (\text{Sub} [f \mid x::[x]]m) - \{x\}, \text{by Theorem 3.1,}
\]

\[
= (\bigcup \{\text{Free} ([f \mid x::[x]]^\infty y) \mid y \in \text{Free} m\} - \{x\},
\]

\[
\text{by the induction hypothesis,}
\]

\[
= ((\bigcup \{\text{Free}([f \mid x::[x]]^\infty y) \mid y \in \text{Free} m, y \neq x\})
\]

\[
\cup \{x\}) - \{x\},
\]

\[
\text{because Free} ([f \mid x::[x]]^\infty x) = \text{Free} [x] = \{x\},
\]

\[
= ((\bigcup \{\text{Free}(f^\infty y) \mid y \in \text{Free} m, y \neq x\})
\]

\[
\cup \{x\}) - \{x\}
\]

\[
= (\text{clashset} <f, x, m> \cup \{x\}) - \{x\}
\]

\[
= \text{clashset} <f, x, m>
\]

\[
= \bigcup \{\text{Free} (f^\infty y) \mid y \in \text{Free} [\lambda x: \tau. m]\}.
\]

Similarly, if \(x \in \text{clashset} <f, x, m>\), then 

\[
\text{Free} (\text{Sub} f [\lambda x: \tau. m])
\]

\[
= \text{Free} [\lambda z: \tau. (\text{Sub} [f \mid x::[z]] m)]
\]

\[
= \text{Free} (\text{Sub} [f \mid x::[z]] m) - \{z\}
\]

\[
= (\bigcup \{\text{Free} ([f \mid x::[z]]^\infty y) \mid y \in \text{Free} m\}) - \{z\}
The next item on the agenda is the consideration of the nature of type checking. To this end, we suppose that in this and the next chapter we are given a fixed function type \( \in G \rightarrow \text{Ob} \ |T| \), which assigns a type to each linguistic constant. For instance, if bool-exp and int-exp are primitive phrase types and true, 2, + \( \in G \), then perhaps

\[
\text{type true} = \text{bool-exp, type 2} = \text{int-exp, type +} = \text{int-exp = int-exp = int-exp}.
\]

Exactly what role does the function type play if we attempt to assign phrase types to all phrases of \( \text{L} \)?

The first difficulty is how do we assign phrase types to expressions containing free occurrences of identifiers. We expect that the type "connected" to \( x \) in \( [\lambda x: \tau. \ell] \) is \( \tau \), but what is the type of \( [x] \)? At first glance, we might be tempted to assign phrase types only to those \( \ell \in \text{L} \) such that \( \text{Free} \ell = \emptyset \). This is entirely inadequate because it prevents us from expressing the type of an expression in terms of the types of its constituents (which possibly contain free
identifiers). A better idea is to assign phrase types to expressions in the context of an assignment of types to identifiers. Thus, the phrase type associated with \([x \ y]\) will be \textit{int-exp} in the context in which \(x\) is assigned \textit{bool-exp} \(\Rightarrow\) \textit{int-exp} and \(y\) is assigned \textit{bool-exp}, whereas \([x \ y]\) will be assigned \(ns = T_{ns}\) (= nonsense) if \(x\) is assigned \textit{bool-exp} and \(y\) is assigned \textit{int-exp}. Note our intended use of the nonsense type: the type checker assigns nonsense to phrases that in the context of a type assignment are meaningless, i.e. not executable, due to mismatched types.

Thus we are led to introduce the poset \(A\), an element of which is a \textit{type assignment}, i.e. a function
\[
\alpha \in F \rightarrow \text{Ob } |T|,
\]
where \(F\) is a finite set of identifiers. Hence
\[
\text{Ob } A = (\text{Ob } |T|)^*.
\]
The partial order on \(A\) is given by
\[
\alpha \leq_A \beta \iff \text{dom } \beta \subseteq \text{dom } \alpha, \text{ and, for all } x \in \text{dom } \beta, \\
\alpha x \leq |T| \beta x.
\]
Intuitively \(\alpha \leq_A \beta\) means \(\alpha\) has more information than \(\beta\), and \(\alpha\) can have more information in two ways: (1) \(\alpha\) can have more components than \(\beta\), and (2) each component of \(\alpha\) can have more information than the corresponding component of \(\beta\). Consistent with this intuition is the observation that the empty type assignment, denoted
\[
\text{emp}_T \in \emptyset \rightarrow \text{Ob } |T|
\]
which contains no information, is the maximal element of \(A\).
A comprehensive type assignment is a function

\[ a \in \text{Id} \to \text{Ob} \, |T| . \]

The collection of comprehensive type assignments can be made into a poset \( A^\infty \) by letting

\[ a \preceq_{A^\infty} b \iff \text{ for all } x \in \text{Id}, \ ax \preceq |T| \ bx . \]

We can write

\[ \text{Ob} \, A^\infty = (\text{Ob} \, |T|)^\infty . \]

Comprehensive type assignments bear roughly the same relationship to type assignments that comprehensive phrase assignments bear to phrase assignments. If \( \alpha \in A \), then there is a canonical \( \alpha^\infty \in A^\infty \) given by

\[ \alpha^\infty x = \begin{cases} \alpha x, & \text{if } x \in \text{dom} \alpha, \\ \text{ns}, & \text{otherwise} . \end{cases} \]

Observe that \( \alpha \preceq_A \beta \) implies \( \alpha^\infty \preceq_{A^\infty} \beta^\infty \).

Our aim is to define and investigate a function

\[ \text{Type} \in |L| \to \text{Ob} \ A \to \text{Ob} \ |T| \]

which assigns to a phrase \( \ell \) in the context of a type assignment \( \alpha \) an intuitively satisfying phrase type \( \text{Type} \ell \alpha \). We expect this function to have four properties. Since a linguistic constant contains no identifiers we expect its type to be independent of the type assignment. This gives the first property.
(Type1) Type $g \alpha = type g$ for all $g \in G$, $\alpha \in A$.

Phrase types of identifiers should be determined by the type assignment, provided the type assignment has enough information in it to do so. This yields the second property.

(Type2) Type $[x] \alpha = \alpha^x x$ for all $x \in Id$, $\alpha \in A$.

The third property says that for an application to make sense in a context, the operator in that context must possess an arrow type, and it must be possible to coerce the type of the operand in that context into the appropriate argument (or parameter) type; then the whole phrase has the appropriate result (or call) type.

(Type3) \[ \begin{align*}
\text{Type } [\ell m] \alpha &= \begin{cases} 
\emptyset, & \text{if Type } \ell \alpha = \tau = \emptyset \text{ and Type } m \alpha \leq \tau. \\
\text{ns}, & \text{otherwise}, \\
\text{for all } \ell, m \in L, \alpha \in A.
\end{cases}
\end{align*} \]

Finally, lambda expressions have procedural types in which the type of the result is computed with an altered type assignment.

(Type4) Type $[\lambda x : \tau. \ell] \alpha = \tau \Rightarrow (\text{Type } \ell [\alpha | x : \tau])$,

for all $x \in Id$, $\tau \in T$, $\ell \in L$, $\alpha \in A$.

Of course, the four properties above suffice to define Type $\ell$ by induction on the structure of $\ell$. However that path fails to uncover fully the role played by $\Lambda$-algebras in type checking. A more fruitful approach begins by defining the $\Lambda$-algebra $E$ whose carrier is

$$|E| = A \to |T|$$,
the collection of functors (i.e. monotone functions) from 
$A$ to $|T|$. The fact that the carrier of $E$ is the collection 
of objects of the poset $A \Rightarrow |T|$ will be exploited in 
Theorem 3.12. The definitions of interpretations of the 
operations on $E$ are contained in the next theorem.

Theorem 3.7: Let $e$ and $f$ be monotone functions from 
$A$ to $|T|$. Let $x \in \text{Id}$ and $\tau \in |T|$. The equations

(1) $E_x \alpha = \alpha^x x$,

(2) $E_{\lambda x : \tau} (e, f) \alpha = \begin{cases} 
\emptyset & \text{if } e\alpha = \tau \Rightarrow \emptyset \text{ and } f\alpha \leq \tau \\
\text{ns} & \text{otherwise},
\end{cases}$

(3) $E_{\lambda x : \tau} e \alpha = \tau \Rightarrow (e \left[ \alpha \mid x : \tau \right])$,

where $\alpha \in A$, define monotone functions from $A$ to $|T|$.

Proof: Suppose $\alpha \preceq_A \beta$.

(1) From $\alpha^\infty \preceq_A \beta^\infty$ we get

$$E_x \alpha = \alpha^x x \preceq \beta^x x = E_x \beta.$$

(2) Suppose $e\beta \Rightarrow \tau \Rightarrow \emptyset$ and $f\beta \leq \tau$, so that 
$E_{\lambda x : \tau} (e, f) \beta = \emptyset$. Since $e\alpha \preceq e\beta$, Theorem 2.2 shows that $e\alpha$
must be a procedural type, say $e\alpha = \tau' \Rightarrow \emptyset'$ where $\tau \leq \tau'$
and $\emptyset' \leq \emptyset$. Then $f\alpha \leq f\beta \leq \tau \leq \tau'$, which implies
$E_{\lambda x : \tau} (e, f) \alpha = \emptyset' \leq \emptyset = E_{\lambda x : \tau} (e, f) \beta$, as desired.

Otherwise $E_{\lambda x : \tau} (e, f) \beta = \text{ns}$, and the inequality
$E_{\lambda x : \tau} (e, f) \alpha \preceq E_{\lambda x : \tau} (e, f) \beta$ is trivially true.
(3) Clearly \([\alpha \mid x : \tau] \leq [\beta \mid x : \tau]\). Thus
\[E \lambda x : \tau \ e \alpha = \tau \Rightarrow (e \ [\alpha \mid x : \tau]) \leq \tau \Rightarrow (e \ [\beta \mid x : \tau]).\]

Note that among the elements of \(|E| = A \rightarrow |T|\) are the constant functions. Thus it makes sense to define the \(\Lambda\)-algebra homomorphism

\[
\text{Type } e \quad L \xrightarrow{\Lambda-\text{Alg}} \quad E
\]

to be the unique homomorphism such that, for all \(g \in G\), \(\alpha \in A\),

\[
\text{Type } g \alpha = \text{type } g.
\]

Theorem 3.8: The \(\Lambda\)-algebra homomorphism Type is the unique function in \(|L| \rightarrow (A \rightarrow |T|)\) such that (Type1), (Type2), (Type3), and (Type4) all hold.

Proof: Trivial. \(\square\)

It is interesting to note that approaching Type as a \(\Lambda\)-algebra homomorphism tells us immediately that Type \(\ell\), for each \(\ell\), is monotone. This would certainly not be obvious if we attempted to define Type by structural induction instead of using the algebraic method we have chosen.

The next theorem and its two corollaries show that only the free identifiers in a phrase affect its type.

Theorem 3.9: Let \(\ell \in L\). For all \(\alpha, \beta \in A\),

\[
\alpha \Downarrow x = \beta \Downarrow x, \quad \text{for all } x \in \text{Free } \ell,
\]

implies

\[
\text{Type } \ell \alpha = \text{Type } \ell \beta.
\]
Proof: We proceed by induction on the structure of $\ell \in L$.

1. Suppose $\ell = g$ where $g \in G$. Then, for all $\alpha, \beta \in A$, Type $g \alpha = \text{type } g = \text{Type } g \beta$.

2. Suppose $\ell = [x]$ where $x \in \text{Id}$. Recalling that Free $[x] = \{x\}$, we let $\alpha, \beta \in A$ be such that

$$\alpha^\circ x = \beta^\circ x.$$

Then

$$\text{Type } [x] \alpha = \alpha^\circ x = \beta^\circ x = \text{Type } [x] \beta.$$

3. Suppose $\ell = [mn]$ where $m, n \in L$. Assume that $\alpha, \beta \in A$ satisfy

$$\alpha^\circ x = \beta^\circ x \text{ for all } x \in (\text{Free } m) \cup (\text{Free } n)$$

By the induction hypothesis

$$\text{Type } m \alpha = \text{Type } m \beta, \text{ and Type } n \alpha = \text{Type } n \beta.$$

From condition (Type3) it is then immediate that

$$\text{Type } [mn] \alpha = \text{Type } [mn] \beta.$$

4. Suppose $\ell = [\lambda x: \tau. m]$ where $x \in \text{Id}$, $\tau \in \tau$, $m \in L$. Recall that Free $[\lambda x: \tau. m] = (\text{Free } m) - \{x\}$. Thus, take $\alpha, \beta \in A$ such that

$$\alpha^\circ y = \beta^\circ y, \text{ for all } y \in (\text{Free } m) - \{x\}.$$

Then

$$[\alpha \mid x : \tau]^\circ y = [\beta \mid x : \tau]^\circ y \text{ for all } y \in \text{Free } m.$$

We conclude by the induction hypothesis that
Type $[\lambda x: \tau.m] \alpha = \tau \rightarrow (\text{Type } m \ [\alpha \mid x: \tau])$

$= \tau \rightarrow (\text{Type } m \ [\beta \mid x: \tau])$

$= \text{Type } [\lambda x: \tau.m] \beta.$

\[\square\]

**Corollary 3.10:** Suppose $\ell \in L$, $\alpha \in A$, $\beta \in A$, and $\text{dom } \alpha = \text{dom } \beta = F$. If

$\alpha x = \beta x$, for all $x \in (\text{Free } \ell) \cap F$,

then

$\text{Type } \ell \alpha = \text{Type } \ell \beta.$

**Proof:** Let $x \in \text{Free } \ell$. If $x \in F$, then

$\alpha^\tau x = \alpha x = \beta x = \beta^\tau x$;

otherwise,

$\alpha^\tau x = \text{ns} = \beta^\tau x$.

Therefore $\text{Type } \ell \alpha = \text{Type } \ell \beta.$ \[\square\]

**Corollary 3.11:** Let $\ell \in L$, $\alpha \in A$, $x \in \text{Id}$, $\tau \in T$. If $x \notin \text{Free } \ell$, then

$\text{Type } \ell \alpha = \text{Type } \ell [\alpha \mid x: \tau].$

**Proof:** Apply Theorem 3.9 with $\beta = [\alpha \mid x: \tau]$. \[\square\]

The final goal of this chapter is to obtain a satisfying description of the type of an expression which is itself the result of a substitution and to investigate some of its consequences. To this end, we will need to define $\text{Type}^*$, a function which is derived from the homomorphism $\text{Type}$ and
which applies to phrase assignments rather than phrases. To avoid confusion, let's review our use of the infixed operator $\rightarrow$. The notation $X \rightarrow Y$ denotes the collection of functions from $X$ to $Y$, unless $X$ and $Y$ are both categories, in which case it denotes the collection of functors from $X$ to $Y$. Therefore $A \rightarrow A$ denotes the set of monotone functions from $A$ to $A$, and $|L|^* \rightarrow (A \rightarrow A)$ designates the set of all functions from the collection of phrase assignments to $A \rightarrow A$. Whereas Type accepts phrases and in an appropriate context produces a type, the function $\text{Type}^*$ accepts phrase assignments and in an appropriate context produces a type assignment. Actually,

$$\text{Type}^* : |L|^* \rightarrow (A \rightarrow A).$$

If $f \in |L|^*$ and $\alpha \in A$, then

$$\text{Type}^* f \alpha \in \text{dom} f \rightarrow \text{Ob} |T|,$$

and

$$\text{Type}^* f \alpha x = \text{Type} (fx) \alpha, \text{ for all } x \in \text{dom} f.$$

We must verify that $\text{Type}^* f$ is monotone. Suppose $\alpha \preceq_A \beta$.

Observe that

$$\text{dom} (\text{Type}^* f \alpha) = \text{dom} f = \text{dom} (\text{Type}^* f \beta).$$

For $x \in \text{dom} f$,

$$\text{Type}^* f \alpha x = \text{Type} (fx) \alpha$$

$$\preceq \text{Type} (fx) \beta, \text{ by the monotonicity of Type} (fx),$$

$$= \text{Type}^* f \beta x.$$

Hence $\text{Type}^* f \alpha \preceq_A \text{Type}^* f \beta$, as desired.
As an aid to contemplating the statement of the next theorem, consider the diagram

\[
\begin{array}{c}
\text{Type}^* f \\
\downarrow \\
\text{Type} \ (\text{Sub} \ f \ \ell) \\
\downarrow \\
|T| \\
\end{array} 
\rightarrow 
\begin{array}{c}
\text{Type} \ell \ , \\
\end{array}
\]

where \( \ell \in L, f \in |L|^* \). In particular, suppose \( \ell = [x] \), where \( x \in \text{Id} \). It is easy to see that if \( f \) maps \( x \) to \( [y] \), where \( y \in \text{Id} \), then the diagram commutes; whereas if \( f = \text{emp} \), then it does not.

**Theorem 3.12:** Let \( \ell \in L \). For all \( f \in |L|^* \),

\[
\text{Type} \ (\text{Sub} \ f \ \ell) \leq_{A \Rightarrow |T|} (\text{Type} \ell) \circ (\text{Type}^* f) ,
\]
and the inequality becomes equality whenever \( \text{Free} \ \ell \subseteq \text{dom} \ f \).

**Proof:** Use induction on the structure of \( \ell \).

(1) Let \( \ell = g \) where \( g \in G \). For all \( f \in |L|^* \), \( \alpha \in A \),

\[
\begin{align*}
\text{Type} \ (\text{Sub} \ f \ g) \ \alpha &= \text{Type} \ g \ \alpha \\
&= \text{type} \ g \\
&= \text{Type} \ g \ (\text{Type}^* f \ \alpha) ,
\end{align*}
\]
as desired.

(2) Let \( \ell = [x] \) where \( x \in \text{Id} \). Suppose \( f \in |L|^* \) and \( \alpha \in A \). If \( x \in \text{dom} \ f \), then

\[
\begin{align*}
\text{Type} \ (\text{Sub} \ f \ [x]) \ \alpha &= \text{Type} \ (fx) \ \alpha , \text{ since } f^\infty x = f x , \\
&= \text{Type}^* f \ \alpha \ x \\
&= (\text{Type}^* f \ \alpha)^\infty x \\
&= \text{Type} \ [x] \ (\text{Type}^* f \ \alpha) .
\end{align*}
\]
If \( x \notin \text{dom} \, f \), then

\[
\text{Type} \left( \text{Sub} \, f \left[ x \right] \right) \alpha = \text{Type} \left[ x \right] \alpha , \quad \text{since} \ f^\infty x = \left[ x \right],
\]

\[
\leq \, \text{ns}
\]

\[
= \left( \text{Type}^* \, f \alpha \right)^\infty x ,
\]

since \( \text{dom} \, \left( \text{Type}^* \, f \alpha \right) = \text{dom} \, f \),

\[
= \text{Type} \left[ x \right] \left( \text{Type}^* \, f \alpha \right)
\]

Hence \( \text{Type} \left( \text{Sub} \, f \left[ x \right] \right) \leq_A \left| \mathcal{T} \right| \left( \text{Type} \left[ x \right] \right) \circ \left( \text{Type}^* \, f \right) \),

and, upon recalling that \( \text{Free} \left[ x \right] = \left[ x \right] \), we have equality
when \( \text{Free} \left[ x \right] \subseteq \text{dom} \, f \).

(3) Let \( L = \left[ mn \right] \) where \( m, n \in L \). Let \( f \in \left| L \right|^* \) and \( \alpha \in A \). We must carry out two computations. First,

\[
\text{Type} \left( \text{Sub} \, f \left[ mn \right] \right) \alpha 
\]

\[
= \text{Type} \left[ \left( \text{Sub} \, f \, m \right) \left( \text{Sub} \, f \, n \right) \right] \alpha 
\]

\[
= \left\{ \begin{array}{ll}
\emptyset & \text{if Type} \left( \text{Sub} \, f \, m \right) \alpha = \tau \Rightarrow \emptyset , \\
\text{Type} \left( \text{Sub} \, f \, n \right) \alpha \leq \tau , & \text{ns otherwise.}
\end{array} \right.
\]

Second,

\[
\text{Type} \left[ mn \right] \left( \text{Type}^* \, f \alpha \right)
\]

\[
= \left\{ \begin{array}{ll}
\emptyset & \text{if Type} \, m \left( \text{Type}^* \, f \alpha \right) = \tau' \Rightarrow \emptyset' , \\
\text{Type} \, n \left( \text{Type}^* \, f \alpha \right) \leq \tau' , & \text{ns otherwise.}
\end{array} \right.
\]

If

\[
\text{Type} \, m \left( \text{Type}^* \, f \alpha \right) = \tau' \Rightarrow \emptyset' \text{ and Type} \, n \left( \text{Type}^* \, f \alpha \right) \leq \tau' ,
\]

then by the induction hypothesis and Theorem 2.2

\[\text{(*)} \quad \text{Type} \left( \text{Sub} \, f \, m \right) \alpha = \tau \Rightarrow \emptyset \leq \tau' \Rightarrow \emptyset' \quad (\text{so that} \ \tau' \leq \tau \text{ and} \ \emptyset \leq \emptyset') ,\]
(*) \[ \text{Type} \, (\text{Sub} \, f \, n) \, \alpha \leq \text{Type} \, n \, (\text{Type}^* \, f \, \alpha) \leq \tau' \]

(*) \[ \leq \tau, \]

and consequently

\[ \text{Type} \, (\text{Sub} \, [mn] \, \alpha) = \theta \]

(*) \[ \leq \theta' = \text{Type} \, [mn] \, (\text{Type}^* \, f \, \alpha); \]

otherwise

(*) \[ \text{Type} \, (\text{Sub} \, [mn] \, \alpha) \leq ns \leq \text{Type} \, [mn] \, (\text{Type}^* \, f \, \alpha). \]

This gives the desired inequality. Suppose in addition that

\[ (\text{Free} \, m) \cup (\text{Free} \, n) = \text{Free} \, [mn] \subseteq \text{dom} \, f. \]

Then we may change all inequalities in lines marked (*) to equalities, yielding the desired equality.

(4) Let \( l = \lambda x : \tau. \, m \) where \( x \in \text{Id}, \, \tau \in T, \, m \in L \). Take any \( f \in |L|^*, \, \alpha \in A \). Let \( X = \text{clashset} \, <f, \, x, \, m> \) and \( z = \text{newid} \, X \). There are two relevant equations. First,

\[ \text{Type} \, (\text{Sub} \, f \, [\lambda x : \tau. \, m]) \, \alpha \]

\[ = \begin{cases} 
\text{Type} \, [\lambda x : \tau. \, (\text{Sub} \, f \mid x : [x] \, m)] \, \alpha \text{ if } x \notin X, \\
\text{Type} \, [\lambda z : \tau. \, (\text{Sub} \, f \mid x : [z] \, m)] \, \alpha \text{ if } x \in X 
\end{cases} \]

\[ = \begin{cases} 
\tau \Rightarrow (\text{Type} \, (\text{Sub} \, f \mid x : [x] \, m) \, [\alpha \mid x : \tau]) \text{ if } x \notin X, \\
\tau \Rightarrow (\text{Type} \, (\text{Sub} \, f \mid x : [z] \, m) \, [\alpha \mid z : \tau]) \text{ if } x \in X. 
\end{cases} \]

Second

\[ \text{Type} \, [\lambda x : \tau. \, m] \, (\text{Type}^* \, f \, \alpha) \]

\[ = \tau \Rightarrow (\text{Type} \, m \, [\text{Type}^* \, f \, \alpha \mid x : \tau]). \]
On the one hand, suppose \( x \notin X \). By the induction hypothesis

\[
\text{Type} \ (\text{Sub} \ [f \mid x::[x]] \ m) \ [\alpha \mid x:\tau] \leq \text{Type} \ m \ (\text{Type}^* \ [f \mid x::[x]] \ [\alpha \mid x:\tau]).
\]

(\dagger)

We will show that

\[
\text{Type} \ m \ \beta = \text{Type} \ m \ \gamma
\]

where

\[
\beta = \text{Type}^* \ [f \mid x::[x]] \ [\alpha \mid x:\tau] \quad \text{and} \quad \gamma = [\text{Type}^* \ f \ \alpha \mid x:\tau]
\]

To see this, note that the domains of \( \beta \) and \( \gamma \) are both equal to \( (\text{dom } f) \cup \{x\} \). If the restrictions of \( \beta \) and \( \gamma \) to \( (\text{Free } m) \cap ((\text{dom } f) \cup \{x\}) \) are equal, then the desired equality follows by Corollary 3.10. So let

\[
y \in (\text{Free } m) \cap ((\text{dom } f) \cup \{x\}). \quad \text{If } y = x, \text{ then}
\]

\[
\beta \ y = \text{Type} \ ([f \mid x::[x]] \ x) \ [\alpha \mid x:\tau]
\]

\[
= \text{Type} \ [x] \ [\alpha \mid x:\tau]
\]

\[
= \tau
\]

\[
= \gamma \ y
\]

If \( y \neq x \), then

\[
\beta \ y = \text{Type} \ ([f \mid x::[x]] \ y) \ [\alpha \mid x:\tau]
\]

\[
= \text{Type} \ (fy) \ [\alpha \mid x:\tau]
\]

\[
= \text{Type} \ (fy) \ \alpha,
\]

by Corollary 3.11 which applies since \( x \notin X \) implies \( x \notin \text{Free } (fy) \),

\[
= \text{Type}^* \ f \ \alpha \ y
\]

\[
= \gamma \ y.
\]

We conclude that \( \text{Type} \ m \ \beta = \text{Type} \ m \ \gamma \). Therefore
Type \((\text{Sub } f [\lambda x: \tau. m]) \alpha\)

\[= \tau \Rightarrow (\text{Type } \text{Sub } [f \mid x:\{x\}] m) [\alpha \mid x: \tau]\]

\((\dagger)\)

\[\leq \tau \Rightarrow (\text{Type } m \beta)\]

\[= \tau \Rightarrow (\text{Type } m \gamma)\]

\[= \text{Type } [\lambda x: \tau. m] (\text{Type}^* f \alpha).\]

On the other hand, suppose \(x \in X\). By the induction hypothesis

\[\text{Type } \text{Sub } [f \mid x:\{z\}] m) [\alpha \mid z: \tau]\]

\((\dagger)\)

\[\leq \text{Type } m (\text{Type}^* [f \mid x:\{z\}] [\alpha \mid z: \tau]).\]

We claim

\[\text{Type } m \delta = \text{Type } m \gamma\]

where \(\gamma\) is as before and

\[\delta = \text{Type}^* [f \mid x:\{z\}] [\alpha \mid z: \tau].\]

Clearly the domains of \(\gamma\) and \(\delta\) are both equal to

\((\text{dom } f) \cup \{x\})\), and by Corollary 3.10 it suffices to show

the equality of the restrictions of \(\gamma\) and \(\delta\) to

\((\text{Free } m) \cap ((\text{dom } f) \cup \{x\})\). Let \(y \in (\text{Free } m) \cap ((\text{dom } f) \cup \{x\}).\)

If \(y = x\), then

\[\delta y = \text{Type } ([f \mid x:\{z\}] x) [\alpha \mid z: \tau]\]

\[= \text{Type } [z] [\alpha \mid z: \tau]\]

\[= \tau\]

\[= \gamma y.\]

If \(y \neq x\), then

\[\delta y = \text{Type } ([f \mid x:\{z\}] y) [\alpha \mid z: \tau]\]

\[= \text{Type } (fy) [\alpha \mid z: \tau]\]

\[= \text{Type } (fy) \alpha,\]

by Corollary 3.11 which applies because

\(z \notin X\) implies \(z \notin \text{Free } (fy)\),
\[ \text{Type}^* \ f \ a \ y \]
\[ = \gamma \ y \].

Much as before, we get

\[ \text{Type} \ (\text{Sub} \ f \ [\lambda x: \tau . m]) \ a \]
\[ = \tau \Rightarrow (\text{Type} \ (\text{Sub} \ [f \ | \ x: [\tau]] \ m) \ [a \ | \ z: \tau]) \]

(+) \[ \leq \tau \Rightarrow (\text{Type} \ m \ \delta) \]
\[ = \tau \Rightarrow (\text{Type} \ m \ \gamma) \]
\[ = \text{Type} \ [\lambda x: \tau . m] \ (\text{Type}^* \ f \ a) \].

Hence

(+) \[ \text{Type} \ (\text{Sub} \ f \ [\lambda x: \tau . m]) \ \leq_{A \Rightarrow |T|} \ (\text{Type} \ [\lambda x: \tau . m]) \circ (\text{Type}^* \ f) \]

Furthermore, if

\[(\text{free} \ m) - \{x\} = \text{free} \ [\lambda x: \tau . m] \subseteq \text{dom} \ f, \]

then every inequality marked (+) may be strengthened to an equality. []

In the next proposition the expression Sub \[ \text{emp}_L \ | \ x: [y] \] \[ \ell \]
has the easy reading "substitute y for x in \ell." It should not be surprising that a satisfying result relating the type of \ell with the type of the phrase arising from substituting y for x in \ell requires the hypothesis that y not occur free in \ell.

Proposition 3.13: Let \( x, y \in \text{Id}, \ \ell \in L, \ \tau \in T, \ a \in A. \) Suppose \( y \notin \text{Free} \ \ell \) and \( m = \text{Sub} \ [\text{emp}_L \ | \ x: [y]] \ \ell \). Then

\[ \text{Type} \ \ell \ [a \ | \ x: \tau] = \text{Type} \ m \ [a \ | \ y: \tau] \]

Proof: Define the phrase assignment \( f \in \text{Free} \ \ell \rightarrow |L| \) by
\[ f z = \begin{cases} [z] & \text{if } z \neq x \\ [y] & \text{if } z = x. \end{cases} \]

By Theorem 3.3, \( \text{Sub } f = \text{Sub } \left[ \text{emp}_L \mid x : [y] \right] \).
Hence \( m = \text{Sub } f \ell \). By Theorem 3.12,

\[ \text{Type } m = (\text{Type } \ell) \circ (\text{Type}^* f). \]

Observe that

\[ \text{Type}^* f [a \mid y : \tau] \in \text{Free } \ell \to |L| \]

is the same as the restriction of \([a \mid x : \tau] &\) to \(\text{Free } \ell\),
because, if \( z \neq x \), then

\[ \text{Type}^* f [a \mid y : \tau] z = \text{Type } (fz) [a \mid y : \tau] \\
= \text{Type } [z] [a \mid y : \tau] \\
= a^\omega z \\
= [a \mid x : \tau]^\omega z, \]

whereas, if \( z = x \), then

\[ \text{Type}^* f [a \mid y : \tau] z = \text{Type } (fx) [a \mid y : \tau] \\
= \text{Type } [y] [a \mid y : \tau] \\
= \tau \\
= [a \mid x : \tau]^\omega z. \]

We may now apply Theorem 3.2 to conclude

\[ \text{Type } m [a \mid y : \tau] = \text{Type } \ell \left( \text{Type}^* f [a \mid y : \tau] \right) \\
= \text{Type } \ell [a \mid x : \tau]. \]

\[ \square \]

Proposition 3.14 explores the connection between
\( \beta \)-reduction and the function \( \text{Type} \). It is used in Chapter IV.
Proposition 3.14: Let \( x \in \text{Id}, \tau \in \Gamma, \) and \( \ell, m \in L. \)

Suppose \( n = \text{Sub} \{ \text{emp}_L \mid x : m \} \ell \) and \( n' = [(\lambda x : \tau. \ell) \m] \).

Then

1. Type \( n \subseteq_{A=|\Gamma|} n' \), and

2. Type \( n \alpha = \text{Type} n' \alpha \) whenever \( \alpha \in A \) and Type \( m \alpha = \tau \).

Proof: (1) Let \( \alpha \in A. \) Define the phrase assignment

\[
\begin{align*}
f_{\alpha} & \in (\text{Free} \ell) \cup (\text{dom} \alpha) \cup \{x\} \rightarrow |L| \\
f_{\alpha} y & = \begin{cases} [y] & \text{if } y \neq x \\
m & \text{if } y = x \end{cases}.
\end{align*}
\]

Using Theorem 3.3, we see \( \text{Sub} f = \text{Sub} \{ \text{emp}_L \mid x : m \} \).

Therefore \( n = \text{Sub} f_{\alpha} \ell. \) Since \( \text{Free} \ell \subseteq \text{dom} f \), Theorem 3.12 gives

Type \( n = (\text{Type} \ell) \circ (\text{Type}^* f_{\alpha}). \)

Observe that for \( y \in (\text{dom} \alpha) \cup \{x\} \)

Type \( \text{Type}^* f_{\alpha} \alpha y = \text{Type} (f_{\alpha} y) \alpha = \begin{cases} \alpha y & \text{if } y \neq x \\
 \text{Type} m \alpha & \text{if } y = x \end{cases}. \)

Hence

Type \( \text{Type}^* f_{\alpha} \alpha \leq_{A} [\alpha \mid x : \text{Type} m \alpha] \)

There are two cases. First suppose Type \( m \alpha \leq \tau. \)

Then

Type \( n \alpha = \text{Type} \ell (\text{Type}^* f_{\alpha}) \)

\[
\leq \text{Type} \ell [\alpha \mid x : \text{Type} m \alpha]
\]

\[
\leq \text{Type} \ell [\alpha \mid x : \tau]
\]

\[
= \text{Type} [(\lambda x : \tau. \ell) \m] \alpha
\]

\[
= \text{Type} n' \alpha.
\]
Second, suppose that it is not true that $\text{Type } m \alpha \leq \tau$.

Then

$$\text{Type } n \alpha \leq ns = \text{Type } [(\lambda x: \tau. \ell) m] \alpha = \text{Type } n' \alpha.$$  

Hence

$$\text{Type } n \leq_{A \Rightarrow | T |} \text{Type } n'.$$

(2) Consider the proof above with the added assumption that $\text{Type } m \alpha = \tau$. It is easy to see that in this case

$$(\text{Type}^* f \alpha)^\infty = [\alpha \mid x: \tau]^\infty.$$  

Thus

$$\text{Type } n \alpha = \text{Type } \ell (\text{Type}^* f \alpha)$$

$$= \text{Type } \ell [\alpha \mid x: \tau]$$

$$= \text{Type } n' \alpha.$$
CHAPTER IV

THE FUNDAMENTAL THEOREM OF SEMANTICS

In this chapter we explore the process of assigning meanings to programs with the aim of uncovering the algebraic roots of programming language semantics.

We continue to assume as given the various entities assumed to exist in the last chapter. Thus, we suppose we are given

1. \( P \), the poset of primitive phrase types, which determines \( T \), the free type algebra generated by \( P \),
2. \( \text{Id} \), the infinite set of identifiers, which with \( T \) determines \( \Lambda \), the language signature,
3. \( G \), the generating set of linguistic constants, which with \( \Lambda \) determines \( L \), the coercive typed \( \lambda \)-calculus,
4. newid \( \epsilon \mid F \mid \to \text{Id} \), which enters into the definition of the substitution function
   \[ \text{Sub} \ \epsilon \mid L \mid^* + \mid L \mid + \mid L \mid, \]
   and
5. type \( \epsilon \mid G \to \text{Ob} \mid T \mid \), which determines
   \[ \text{Type} \ \epsilon \mid L \mid \to (A \to \mid T \mid), \] the type-checking homomorphism.
Furthermore, we assume we are given

(6) $K$, a type algebra derived in the canonical way from its carrier $|K|$, which is assumed to be a Cartesian closed category - hence,

$$K_{\text{ns}} = |K|_{\text{term}} \quad \text{and} \quad K_{\Rightarrow} = |K|_{\Rightarrow},$$

(7) $\text{mng} \in \mathcal{P} \rightarrow |K|$, a functor, which gives rise by Theorem 2.2 to a type algebra homomorphism

$$\text{mng} \in T \xrightarrow{\text{Type Alg}} K,$$

and

(8) a basic semantic function

$$\text{semf} \in G \rightarrow \text{Ar} |K|$$

such that, for all $g \in G$,

$$\text{semf} \ g \in \prod_{\phi} \text{emp} |K| \xrightarrow{\ |K|} \text{mng} \ (\text{type} \ g).$$

Of course, we must give some intuitively satisfying justification for calling something with the properties of semf a semantic function. Suppose for a moment that $|K| = \text{Set}$. The idea is that, for $g \in G$, semf $g$ gives a proper meaning for $g$, i.e. an element of some set. The set to which semf $g$ belongs is given by the typing function. A first guess would be

$$\text{semf} \ g \in \text{mng} \ (\text{type} \ g).$$

If all the Cartesian closed categories of use had objects with a set-like structure, then this might suffice as a basis
for generalization. However, other kinds of Cartesian closed categories, namely functor categories, turn out to play a crucial role in programming language semantics. Thus we must reformulate the condition \( \text{semf } g \in \text{Mng (type g)} \). Let \( T = \{t\} \) be a terminal object of \( \text{Set} \). The elements of any set \( X \) are in one-to-one correspondence with the functions \( T \rightarrow X \). So we are led to revise our notion of what a proper meaning of \( g \) is — no longer is it an element of \( \text{Mng (type g)} \), but rather it is a method of choosing an element of \( \text{Mng (type g)} \). More precisely, we want

\[
\text{semf } g \in T \xrightarrow{\text{Set}} \text{Mng (type g)}.
\]

We have been a bit vague about what terminal object of \( \text{Set} \) should be used as \( T \). In a sense it is immaterial, but for reasons of subsequent elegance it is best to use \( T = \prod_{\phi} \text{emp}_{\text{Set}} \), the set consisting of the unique 0-tuple. Our motivation for (8) is now complete.

It is significant that we have come to the point of view that a semantic function assigns to a phrase a morphism of a Cartesian closed category. The central question is how does

\[
\text{semf } \varepsilon \in G \rightarrow \text{Ar } |K|
\]

determine a semantic function

\[
\text{Semf } \varepsilon |L| \rightarrow ?
\]

applicable to all phrases of \( L \). The appropriate codomain for \( \text{Semf} \) is still in question. It is not \( \text{Ar } |K| \) for much
the same reason that the codomain of Type is not \( \text{Ob}\ |\mathcal{T}| \).
Just as the type of an expression cannot be determined except in the context of a type assignment, the meaning of a phrase requires for its complete determination something that yields an environment, i.e. an assignment of meanings to free identifiers.

Environments are certain product objects in \( |\mathcal{K}| \); they are products of collections of objects indexed by finite sets of identifiers. Environments are produced from type assignments by application of the functor

\[
\text{Env} \in A \mapsto |\mathcal{K}|
\]

which is defined on objects of \( A \) by

\[
\text{Env} \alpha = \prod_{\text{dom} \alpha} (\text{Mng} \circ \alpha).
\]

Suppose \( \alpha \leq_A \beta \). Then

\[
\text{Env} (\alpha \leq_A \beta) \in \text{Env} \alpha \xrightarrow{|\mathcal{K}|} \text{Env} \beta
\]

is the unique morphism such that for all \( x \in \text{dom} \beta \) the diagram

\[
\begin{array}{ccc}
\prod_{\text{dom} \alpha} (\text{Mng} \circ \alpha) & \xrightarrow{\text{proj}_{\text{dom} \alpha}^x} & \text{Mng} (\alpha \times x) \\
\text{Env} (\alpha \leq_A \beta) & \downarrow & \downarrow \\
\prod_{\text{dom} \beta} (\text{Mng} \circ \beta) & \xrightarrow{\text{proj}_{\text{dom} \beta}^x} & \text{Mng} (\beta \times x)
\end{array}
\]

commutes. The verification that \( \text{Env} \) is indeed a functor is routine.
Note that

$$\text{Env} \, \text{emp}_T = \Pi_\phi (\text{Mng} \circ \text{emp}_T) = \Pi_\phi \text{emp} |_K|.$$ 

Thus, the basic semantic function is such that for $g \in G$

$$\text{semf} \, g \in \text{Env} \, \text{emp}_T |_K| \rightarrow \text{Mng} \, (\text{type} \, g).$$

We call $\text{Env} \, \text{emp}_T$ the empty environment. In a sense it is the environment most appropriate to a phrase with no free identifiers, such as a linguistic constant.

It is now clear that the meaning of an expression $\lambda$ should be calculated in the context of a type assignment $\alpha$, which yields an environment $\text{Env} \, \alpha$, and it should ultimately reside in $\text{Mng} \, (\text{Type} \, \lambda \, \alpha)$. Hence we expect $\text{Semf}$ to be found among those functions

$$S : |L| \rightarrow \text{Ob} \, A \rightarrow \text{Ar} |_K|$$

(thus, $S$ assigns to a phrase $\lambda$ an $(\text{Ob} \, A)$-indexed collection $S \, \lambda$ of arrows of $|_K|$) such that

(Semf0) for all $\lambda \in L$, $\alpha \in A$

$$S \, \lambda \, \alpha \in \text{Env} \, \alpha \rightarrow \text{Mng} \, (\text{Type} \, \lambda \, \alpha).$$

In particular we now know the codomain of $\text{Semf}$:

$$\text{Semf} : |L| \rightarrow \text{Ob} \, A \rightarrow \text{Ar} |_K|.$$

Some elements of $\text{Ob} \, A \rightarrow \text{Ar} |_K|$ are particularly nice; they are natural transformations, i.e. arrows of $A \rightarrow |_K|$. Indeed, we shall see that for each $\lambda \in L$, $\text{Semf} \, \lambda$ will be a natural transformation:
\[ \text{Semf } \in \text{Env} \xrightarrow{\text{A} \mapsto |\mathcal{K}|} \text{Mng }^\circ (\text{Type } \alpha). \]

Of course, the class of functions to which Semf belongs should have some other properties in addition to (Semf0) above. For instance a semantic function should give to a linguistic constant a meaning independent of the environment. This is stated more precisely as

(Semf1) for all \( g \in G, \alpha \in A \)

\[ \text{Env } \alpha \xrightarrow{S g \alpha} \text{Mng (type } g) \]

\[ \text{Env } \text{emp } \mathcal{K} \]

commutes, where \( u_\alpha \) is the unique arrow to the empty environment.

We expect identifiers to pick the right meaning out of an environment when this is possible. Thus,

(Semf2) for all \( x \in \text{Id}, \alpha \in A \)

\[ S \|x\| \alpha = \begin{cases} \text{proj} \text{dom } x \in \text{Env } \alpha \xrightarrow{|\mathcal{K}|} \text{Mng}(ax) & \text{if } x \in \text{dom } \alpha, \\ u_\alpha \in \text{Env } \alpha \xrightarrow{|\mathcal{K}|} \text{Mng ns} & \text{otherwise.} \end{cases} \]

In the case of an application we want to coerce the operand into a meaning of the right type and then apply the operator, provided there is no error due to mismatched types. This gives
(Semf3) for all $\ell, m \in L, \alpha \in A$

$\begin{array}{c}
\text{Env} \alpha \\
\downarrow \\
\llbracket \ell m \rrbracket \alpha
\end{array} \rightarrow Mng \theta$

$\begin{array}{c}
\langle \text{Semf } \ell \alpha, \\
\text{Semf } m \alpha \rangle
\downarrow \\
1 \times Mng(\tau' \leq \tau)
\end{array}$

$\text{Ap}(Mng \tau, Mng \theta)$

$\text{commutes, whenever}$

Type $\ell \alpha = \tau \Rightarrow \theta$, and Type $m \alpha = \tau' \leq \tau$,

whereas

$\llbracket \ell m \rrbracket \alpha = u_{\alpha} \in \text{Env} \alpha \\
\downarrow |K|
\rightarrow Mng ns$,

otherwise.

There is a fourth condition connected with the semantics of $\lambda$-expressions, but its statement requires some preliminary groundwork.

Let $x \in \text{Id}$. For each $\alpha \in A, \tau \in \mathcal{T}$ we seek to define

$P_x(\alpha, \tau) \in (\text{Env} \alpha \times (Mng \tau)) \rightarrow \text{Env} [\alpha | x : \tau]$.

The idea is that $P_x(\alpha, \tau)$ is the obvious isomorphism when $x \not\in \text{dom} \alpha$, and $P_x(\alpha, \tau)$ changes the meaning object associated with $x$ from $Mng(\alpha x)$ to $Mng \tau$ when $x \in \text{dom} \alpha$. The morphism $P_x(\alpha, \tau)$ has an important role to play in the treatment of the semantics of identifier binding. Denote by

$\tau_\alpha \in (\text{Env} \alpha \times (Mng \tau)) \\
\downarrow |K|
\rightarrow \text{Env} \alpha$
the projection onto the first component. For each 
y \in \text{dom} [\alpha \mid x : \tau] \text{ define}

\[ P_{x,\alpha,\tau} y \in (\text{Env} \ \alpha) \times (\text{Mng} \ \tau) \rightarrow \text{Mng} \ ([\alpha \mid x : \tau] y) \]

as follows:

(1) if \( y = x \), then

\[ P_{x,\alpha,\tau} x \in (\text{Env} \ \alpha) \times (\text{Mng} \ \tau) \rightarrow \text{Mng} \ \tau \]

is the projection onto the second component,

(2) if \( y \neq x \), then

\[ P_{x,\alpha,\tau} y \in (\text{Env} \ \alpha) \times (\text{Mng} \ \tau) \rightarrow \text{Mng} \ (\alpha y) \]

is the composition

\[ (\text{Env} \ \alpha) \times (\text{Mng} \ \tau) \xrightarrow{\pi_\alpha} \text{Env} \ \alpha \xrightarrow{\text{proj}_\text{dom} \ \alpha} \text{Mng} \ (\alpha y) . \]

Now take \( P_x (\alpha,\tau) \) to be the unique morphism such that

\[ (\text{Env} \ \alpha) \times (\text{Mng} \ \tau) \]

\[ \xrightarrow{P_x (\alpha,\tau)} \]

\[ \xrightarrow{\text{proj}_\text{dom} [\alpha \mid x : \tau]} \text{Mng} \ ([\alpha \mid x : \tau] y) \]

commutes for all \( y \in \text{dom} [\alpha \mid x : \tau] \).

The following technical lemma is used in the proof of the proposition that follows it.

**Lemma 4.1:** Suppose \( x \in \text{Id} \), \( \alpha \preceq_A \beta \), and \( \tau \preceq_\tau \theta \).

For all \( y \in \text{dom} [\beta \mid x : \tau] \), the diagram
\[(\text{Env } \alpha) \times (\text{Mng } \tau) \xrightarrow{\text{P}_x, \alpha, _\tau Y} \text{Mng } ([\alpha \mid x: \tau] y)
\]
\[\text{Env}(\alpha \leq \beta) \times \text{Mng}(\tau \leq \theta) \quad \text{Mng}([\alpha \mid x: \tau] y \leq [\beta \mid x: \theta] y) \]
\[(\text{Env } \beta) \times (\text{Mng } \theta) \xrightarrow{\text{P}_x, \beta, _\theta Y} \text{Mng } ([\beta \mid x: \theta] y)\]

commutes.

**Proof:** Suppose \(y = x\). Then the diagram becomes

\[(\text{Env } \alpha) \times (\text{Mng } \tau) \xrightarrow{} \text{Mng } \tau
\]
\[\text{Env}(\alpha \leq \beta) \times \text{Mng}(\tau \leq \theta) \quad \text{Mng } (\tau \leq \theta) ,
\]
\[(\text{Env } \beta) \times (\text{Mng } \theta) \xrightarrow{} \text{Mng } \theta
\]

where the horizontal arrows are projections onto the second component; this diagram commutes by the definition of the product morphism \(\text{Env} (\alpha \leq \beta) \times \text{Mng} (\tau \leq \theta)\).

Suppose \(y \neq x\). By decomposing the horizontal arrows we get

\[\xrightarrow{\pi_\alpha} \xrightarrow{\text{proj}_{\text{dom } \alpha} Y} \text{Mng } (\alpha y)\]
\[\text{Env}(\alpha \leq \beta) \quad \text{Env}(\alpha \leq \beta) \quad \text{Mng}(\alpha y \leq \beta y)\]
\[\xrightarrow{\pi_\beta} \xrightarrow{\text{proj}_{\text{dom } \beta} Y} \text{Mng } (\beta y)\]

The left-hand square commutes by the definition of the product morphism \(\text{Env} (\alpha \leq \beta) \times \text{Mng} (\tau \leq \theta)\), and the right-hand square commutes by the definition of the functor Env. \(\Box\)

**Proposition 4.2:** Let \(x \in \text{Id}\). Then \(\text{P}_x\) is a natural transformation:
\[
P_x \in (\text{Env} \to \times \text{Mng} \to) \xrightarrow{\lambda x: \top \Rightarrow \top} \text{Env} \to[-x;\top].
\]

**Proof:** Suppose \( \alpha \leq_A \beta \) and \( \tau \leq_T \theta \). We must verify that

\[
\begin{array}{ccc}
(\text{Env} \alpha) \times (\text{Mng} \tau) & \xrightarrow{P_x(\alpha, \tau)} & \text{Env} [\alpha \mid x: \tau] \\
\downarrow \text{Env}(\alpha \leq \beta) \times \text{Mng}(\tau \leq \theta) & & \downarrow \text{Env}([\alpha \mid x: \tau] \leq [\beta \mid x: \theta]) \\
(\text{Env} \beta) \times (\text{Mng} \theta) & \xrightarrow{P_x(\beta, \theta)} & \text{Env} [\beta \mid x: \theta]
\end{array}
\]

is a commutative diagram. We will use the fact that

\[
\text{Env} [\beta \mid x: \theta] = \Pi_{\text{dom}[\beta \mid x: \theta]} (\text{Mng} \circ [\beta \mid x: \theta])
\]

together with the universal mapping property for products. First, for each \( y \in \text{dom}[\beta \mid x: \theta] \), consider the diagram

\[
\begin{array}{ccc}
(\text{Env} \alpha) \times (\text{Mng} \tau) & \xrightarrow{P_x(\alpha, \tau)} & \text{Mng} ([\alpha \mid x: \tau] y) \\
\downarrow P_x(\alpha, \tau) & & \downarrow \text{proj}_{\text{dom}[\alpha \mid x: \tau] y} \\
\text{Env} [\alpha \mid x: \tau] & \xrightarrow{\text{proj}_{\text{dom}[\alpha \mid x: \tau] y}^{-1}} & \text{Mng} ([\alpha \mid x: \tau] y) \\
\downarrow \text{Env}([\alpha \mid x: \tau] \leq [\beta \mid x: \tau]) & & \downarrow \text{Mng}([\alpha \mid x: \tau] y \leq [\beta \mid x: \theta] y).
\end{array}
\]

The inner diagram on the top commutes by the definition of \( P_x(\alpha, \tau) \) and commutativity of the inner diagram on the bottom follows from the definition of Env; thus, the outer compositions are equal. Second, for each \( y \in \text{dom}[\beta \mid x: \theta] \), consider the diagram
Here, the top inner diagram commutes by the preceding lemma, and the lower inner diagram commutes by the definition of $P_x(\beta, \theta)$. Consequently the outer diagram commutes. By the universal mapping property for products, the left-hand vertical compositions in the two diagrams must be equal. □

Now let's return to the statement of the last property that we deem desirable for the collection of those functions

$$S: |L| \to \text{Ob}A + \text{Ar} |K|$$

of which Semf is a member. We expect the meaning of a $\lambda$-expression in the context of a type assignment to be the abstraction of a morphism connected with the meaning of the body of the $\lambda$-expression in the context of the type assignment altered to take into account the binding of an identifier to a type. Therefore,

(Semf4) for all $x \in \text{Id}$, $\tau \in T$, $\lambda \in L$, $\alpha \in A$,

$$S[\lambda x: \tau. \lambda] \alpha \in \text{Env} \alpha \mid_{K} \to \text{Mng} (\tau \Rightarrow (\text{Type} \lambda \mid \alpha \mid x: \tau))$$

is the unique morphism such that
Our investigation of Semf involves two \( \Lambda \)-algebras \( U \) and \( V \), where \( U \) is a subalgebra of \( V \). Take the carrier of \( V \) to be the set of ordered pairs \( (T, F) \) such that

1. \( T \in A \rightarrow |T| \) (hence, \( T \) is monotone), and
2. \( F \) is an \( (\text{Ob } A) \)-indexed collection of arrows of \( |K| \) satisfying for all \( a \in A \)

\[
F \cdot a \in \text{Env } a \xrightarrow{|K|} \text{Mng } (T \cdot a)
\]

The carrier of the subalgebra \( U \) of \( V \) consists of those \( (T, F) \in |V| \) such that \( F \) is a natural transformation:

\[
F \in \text{Env } \xrightarrow{A \rightarrow |K|} \text{Mng } \circ T.
\]

(Thus, the carrier of \( U \) is the collection of objects of the comma category \( (\text{Env } \downarrow A \Rightarrow \text{Mng}) \) where

\[
A \Rightarrow \text{Mng } \varepsilon (A \Rightarrow |T|) \times (A \Rightarrow |K|)
\]

is the functor induced by

\[
\text{Mng } \varepsilon T \xrightarrow{\text{TypeAlg}} K.
\]

The significance of this observation is not completely clear.)

Recall that \( A \rightarrow |T| \) is the carrier of the \( \Lambda \)-algebra \( E \). We turn now to the interpretations in \( V \) of the operators
in $\Lambda$.

Let $x \in \text{Id}$. Let

$$V_x = \langle E_x, F_x \rangle,$$

where

$$F_x \alpha \in \text{Env} \xrightarrow{K} \text{Mng} (E_x \alpha), \ \alpha \in A,$$

will be defined shortly. Upon recalling that

$$\varepsilon_x \alpha = x^\infty = \begin{cases} \alpha x & \text{if } x \in \text{dom } \alpha \\ \text{ns} & \text{otherwise} \end{cases},$$

we let

$$F_x \alpha = \begin{cases} \text{proj}_{\text{dom} \alpha} x \in \text{Env} \xrightarrow{K} \text{Mng} (\alpha x) & \text{if } x \in \text{dom } \alpha, \\ u_\alpha \in \text{Env} \xrightarrow{K} \text{Mng} \text{ ns} & \text{otherwise}. \end{cases}$$

Let $\langle T, F \rangle$ and $\langle T', F' \rangle$ be in $|V|$. Let

$$V_{ap} \langle \langle T, F \rangle, \langle T', F' \rangle \rangle = \langle T_{ap}, F_{ap} \rangle$$

where

$$T_{ap} = \varepsilon_{ap} \langle T, T' \rangle$$

and

$$F_{ap} \alpha \in \text{Env} \xrightarrow{K} \text{Mng} (T_{ap} \alpha), \ \alpha \in A,$$

will be defined below. Note that, for $\alpha \in A$,

$$T_{ap} \alpha = \begin{cases} \emptyset & \text{if } T\alpha = \tau \emptyset \text{ and } T'\alpha \leq \tau, \\ \text{ns} & \text{otherwise.} \end{cases}$$
This leads to two cases in the definition of $F_{ap}$. If $\alpha \in A$, $T\alpha = \tau \Rightarrow \theta$, and $T'\alpha \leq \tau$, then let $F_{ap} \alpha$ be defined by the commutative diagram

$$
\begin{array}{ccc}
\text{Env } \alpha & \xrightarrow{F_{ap} \alpha} & \text{Mng } \theta \\
\downarrow \langle F\alpha, F'\alpha \rangle & & \downarrow \text{Ap}(\text{Mng } \tau, \text{Mng } \theta) \\
(\text{Mng } (T\alpha)) \times (\text{Mng } (T'\alpha)) & \xrightarrow{1 \times \text{Mng } (T'\alpha \leq \tau)} & (\text{Mng } (\tau \Rightarrow \theta)) \times (\text{Mng } \tau)
\end{array}
$$

For other $\alpha \in A$, $F_{ap} \alpha$ is the unique arrow to the nonsense object:

$$
F_{ap} \alpha = u_{\alpha} \in \text{Env } \alpha \xrightarrow{\llbracket K \rrbracket} \text{Mng } \text{ns}
$$

Let $x \in \text{Id}$, $\tau \in T$, $(T, F) \in \mid V \mid$. Let

$$
V_{\lambda x: \tau} (T, F) = (E_{\lambda x: \tau} T, F_{\lambda x: \tau})
$$

where the definition of

$$
F_{\lambda x: \tau} \alpha \in \text{Env } \alpha \xrightarrow{\llbracket K \rrbracket} \text{Mng } (E_{\lambda x: \tau} T \alpha), \alpha \in A,
$$

follows. Recall that

$$
E_{\lambda x: \tau} T \alpha = \tau \Rightarrow (T[\alpha \mid x: \tau]).
$$

For $\alpha \in A$, let $F_{\lambda x: \tau} \alpha$ be the unique morphism in $\mid K \mid$ such that

$$
\begin{array}{ccc}
(\text{Env } \alpha) \times (\text{Mng } \tau) & \xrightarrow{(F_{\lambda x: \tau} \alpha) \times 1} & (\text{Mng } (\tau \Rightarrow (T[\alpha \mid x: \tau]))) \times (\text{Mng } \tau) \\
\downarrow P_x(\alpha, \tau) & & \downarrow \text{Ap}(\text{Mng } \tau, \text{Mng } (T[\alpha \mid x: \tau])) \\
\text{Env } [\alpha \mid x: \tau] & \xrightarrow{P[\alpha \mid x: \tau]} & \text{Mng } (T[\alpha \mid x: \tau])
\end{array}
$$
commutes. This completes the definition of the $\Lambda$-algebra $V$.

**Theorem 4.3:** $U$ is a subalgebra of the $\Lambda$-algebra $V$.

**Proof:** We must show that $|U|$ is closed under the interpretations in $U$ of the operators in $\Lambda$.

Let $x \in \text{Id}$, and consider $V_x = \{\xi_x, F_x\}$. We will show $V_x \in |U|$ by demonstrating that $F_x$ is a natural transformation. Suppose $\alpha \leq_A \beta$, so that $\alpha^\omega x \leq \beta^\omega x$. Consider the diagram

$$
\begin{array}{ccc}
\text{Env} \alpha & \xrightarrow{F_x^\alpha} & \text{Mng} (\alpha^\omega x) \\
\downarrow \text{Env}(\alpha \leq \beta) & & \downarrow \text{Mng}(\alpha \leq \beta x) \\
\text{Env} \beta & \xrightarrow{F_x^\beta} & \text{Mng} (\beta^\omega x)
\end{array}
$$

If $x \notin \text{dom} \beta$, then $\beta^\omega x = \text{ns}$ and the fact that $\text{Mng} (\beta x)$ is terminal gives commutativity of the diagram. If $x \in \text{dom} \beta$, then also $x \in \text{dom} \alpha$, $\alpha x = \alpha^\omega x$, $\beta x = \beta^\omega x$, and commutativity follows from the definition of Env $(\alpha \leq \beta)$.

Let $(T, F)$ and $(T', F')$ be in $|U|$, and consider

$$V_{ap} \langle (T, F), (T', F') \rangle = \langle T_{ap}, F_{ap} \rangle.$$

We wish to show this is in $|U|$, i.e. to show $F_{ap}$ is natural. To do this let $\alpha \leq_A \beta$. Consider what happens when $T\beta = \tau = \emptyset$ and $T'\beta \leq \tau$. Then we can write $T\alpha = \hat{\tau} = \hat{\emptyset}$ where $T'\alpha \leq T'\beta \leq \tau \leq \hat{\tau}$ and $\hat{\emptyset} \leq \emptyset$. We wish to demonstrate the commutativity of the outer square in the diagram.
The interior diagrams labelled (1) and (2) commute by definition, and the square labelled (3) commutes by the naturality of $F$ and $F'$. Let

\[ X = \text{Mng } \tau, \quad \hat{X} = \text{Mng } \hat{\tau}, \quad \gamma = \text{Mng } (\tau \leq \hat{\tau}), \]
\[ Y = \text{Mng } \theta, \quad \hat{Y} = \text{Mng } \hat{\theta}, \quad \delta = \text{Mng } (\hat{\theta} \leq \theta), \]
\[ Z = \text{Mng } (T' \beta), \quad \hat{Z} = \text{Mng } (T' \alpha), \quad \eta = \text{Mng } (T' \alpha \leq T' \beta), \]
\[ \kappa = \text{Mng } (T' \beta \leq \tau) \]

Then the interior diagram labelled (4) becomes
Equivalently, we have

\[
\begin{align*}
(\hat{X} \Rightarrow \hat{Y}) \times \hat{Z} \xrightarrow{1 \times \eta} (\hat{X} \Rightarrow \hat{Y}) \times \hat{X} \xrightarrow{1 \times \kappa} (\hat{X} \Rightarrow \hat{Y}) \times \hat{X} \xrightarrow{1 \times \gamma} (\hat{X} \Rightarrow \hat{Y}) \times \hat{X} \xrightarrow{\text{Ap}(\hat{X}, \hat{Y})} \hat{Y} \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
(\gamma \Rightarrow \delta) \times \eta \xrightarrow{(5)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(6)} (\gamma \Rightarrow \delta) \times 1 \xrightarrow{(7)} \delta \\
\end{align*}
\]

Squares (5) and (6) obviously commute. Diagram (7) commutes by the definition of $\gamma \Rightarrow \delta$. Hence diagram (4) commutes, yielding commutativity of the original diagram. In other cases $T\beta = n\beta$, which automatically gives commutativity of

\[
\begin{align*}
\text{Env} \alpha \xrightarrow{F_{\alpha \beta}} \text{Mng} (T\alpha) \\
\text{Env}(\alpha \leq_A \beta) \xrightarrow{F_{\alpha \beta}} \text{Mng}(T(\alpha \leq_A \beta)) \\
\text{Env} \beta \xrightarrow{F_{\alpha \beta}} \text{Mng} (T\beta) \\
\end{align*}
\]

This concludes the proof that $F_{\alpha \beta}$ is a natural transformation.

Let $x \in \text{Id}, \tau \in T$, $(T,F) \in |U|$. Let

\[
\nu_{\lambda x : \tau} (T,F) = \langle \text{Env}_{\lambda x : \tau} T, F_{\lambda x : \tau} \rangle ;
\]

we aim to show $F_{\lambda x : \tau}$ is a natural transformation.

Suppose $\alpha \leq_A \beta$. Since $\text{Mng}$ is a type algebra homomorphism the morphisms
\[ \text{Mng} \left( \left( \tau \Rightarrow (T \cdot \alpha \mid x : \tau) \right) \right) \leq \left( \tau \Rightarrow (T \cdot \beta \mid x : \tau) \right) \]

and

\[ l_{\text{Mng}} \tau \xrightarrow{\text{K}} \text{Mng} \left( (T \cdot \alpha \mid x : \tau) \right) \leq (T \cdot \beta \mid x : \tau) \]

are equal. Therefore our aim is to prove that

\[ \text{Env} \alpha \xrightarrow{F_{\lambda x : \tau} \alpha} \text{Mng} \tau \Rightarrow \text{Mng} (T \cdot \alpha \mid x : \tau) \]

\[ \text{Env}(\alpha \leq \beta) \]

\[ \text{Env} \beta \xrightarrow{F_{\lambda x : \tau} \beta} \text{Mng} \tau \Rightarrow \text{Mng} (T \cdot \beta \mid x : \tau) \]

commutes. The method of proof is based on the universal mapping property \((\text{Cart3})\), with \(X = \text{Env} \alpha, Y = \text{Mng} \tau, Z = \text{Mng} (T \cdot \beta \mid x : \tau)\). Consider the diagram

\[ \begin{array}{c}
\text{Env}(\alpha) \times (\text{Mng} \tau) \\
\downarrow (F_{\lambda x : \tau} \alpha) \times 1 \\
\approx (\text{Mng} \tau \Rightarrow \text{Mng} (T \cdot \alpha \mid x : \tau)) \times (\text{Mng} \tau) \\
\end{array} \]

\[ (1) \]

\[ \begin{array}{c}
(F_{\lambda x : \tau} \alpha) \times 1 \\
\approx (\text{Mng} \tau \Rightarrow \text{Mng} (T \cdot \alpha \mid x : \tau)) \times (\text{Mng} \tau) \\
\end{array} \]

\[ (2) \]

\[ \begin{array}{c}
\text{Env}(\alpha \mid x : \tau) \\
\downarrow F_{\lambda x : \tau} \alpha \\
\approx \text{Mng} (T \cdot \alpha \mid x : \tau) \\
\end{array} \]

\[ (3) \]

\[ \begin{array}{c}
\text{Env}(\alpha \mid x : \tau) \\
\downarrow < [\beta \mid x : \tau] \\
\approx \text{Mng} (T \cdot \beta \mid x : \tau) \\
\end{array} \]

\[ (4) \]

\[ \begin{array}{c}
\text{Mng} (\ldots) \\
\end{array} \]

\[ \begin{array}{c}
\text{Ap} (\text{Mng} \tau, \text{Mng} (T \cdot \beta \mid x : \tau)) \\
\end{array} \]
The four small diagrams commute for the following reasons:

1. \(- \times 1\) is a functor,
2. naturality of Ap in its second argument,
3. \(F\) is a natural transformation,
4. definition of \(F_{\lambda x : \tau} \alpha\).

Thus the whole diagram commutes. Next, consider the diagram

\[
\begin{array}{c}
(\text{Env}\alpha) \times (\text{Mng}\tau) \xrightarrow{(F_{\lambda x : \tau} \beta) \times (\text{Env}(\alpha \leq \beta)) \times 1} (\text{Mng}\tau \Rightarrow \text{Mng}(T[\beta | x : \tau])) \\
\downarrow P_x(\alpha, \tau) \xrightarrow{(\text{Env}(\alpha \leq \beta) \times 1)} (\text{Env}\beta) \times (\text{Mng}\tau) \xrightarrow{(F_{\lambda x : \tau} \beta) \times 1} (\text{Ap}(\text{Mng}\tau, \text{Mng}(T[\beta | x : \tau]))) \\
\downarrow \text{Env}[\alpha | x : \tau] \xrightarrow{P_x(\beta, \tau)} \text{Env}[\beta | x : \tau] \xrightarrow{F[\beta | x : \tau]} \text{Mng}(T[\beta | x : \tau]).
\end{array}
\]

The reasons for the commutativity of the small diagrams is as follows:

5. \(- \times 1\) is a functor,
6. definition of \(F_{\lambda x : \tau} \beta\),
7. naturality of \(P_x\).

So the entire diagram commutes. We finally apply the universal mapping property (Cart3) to get commutativity of the square in question. ☐

The condition (Semfl) suggests that we should use the function
\[ \text{semf} \in G \rightarrow \text{Ar } |K| \]

to create

\[ \text{semf}' \in G \rightarrow \text{Ob } A \rightarrow \text{Ar } |K| , \]

where \( \text{semf}' \) is defined for \( g \in G, \alpha \in A \) by the commutative diagram

\[ \begin{array}{ccc}
\text{Env } \alpha & \xrightarrow{\text{semf}' g \alpha} & \text{Mng } (\text{Type } g \alpha) \\
\downarrow{u_\alpha} & \downarrow{\text{semf } g} & \downarrow{\text{Env } \text{emp}_T} \\
\end{array} \]

Now define

\[ \text{sf} \in G \rightarrow |V| \]

by

\[ \text{sf } g = \langle \text{Type } g, \text{semf}' g \rangle \]

for \( g \in G \). Extend \( \text{sf} \) to get

\[ \text{sf} \in L \xrightarrow{\Lambda-\text{Alg}} V, \]

the unique \( \Lambda \)-algebra homomorphism such that

\[ \text{sf } g = \text{sf } g \quad \text{for all } g \in G . \]

Let

\[ \text{Pr}_1 \in |V| \rightarrow |E| , \quad \text{Pr}_2 \in |V| \rightarrow \text{Ob } A \rightarrow \text{Ar } |K| \]

be the functions which select the components of the elements of
Thus $\Pr_1(T, F) = T$ and $\Pr_2(T, F) = F$.

**Proposition 4.4:** The function $\Pr_1$ is a $\Lambda$-algebra homomorphism:

$$\Pr_1 \in V \xrightarrow{\Lambda-Alg} E.$$

**Proof:** Clear. []

**Corollary 4.5:** The diagram

$$
\begin{array}{ccc}
\text{Type} & \rightarrow & E \\
L & \rightarrow & \mathcal{E} \\
\text{SF} & \downarrow & \text{Pr}_1 \\
V & \rightarrow & \\
\end{array}
$$

of $\Lambda$-algebra homomorphisms is commutative.

**Proof:** For all $g \in G$

$$\text{Type } g = \Pr_1(\text{SF } g).$$

Define the semantic function

$$\text{Sem} \in |L| \rightarrow \text{Ob } A \rightarrow \text{Ar } |K|$$

by the commutative diagram

$$
\begin{array}{ccc}
\text{Sem} & \rightarrow & (\text{Ob } A \rightarrow \text{Ar } |K|) \\
|L| & \rightarrow & \\
\text{SF} & \downarrow & \text{Pr}_2 \\
|V| & \rightarrow & \\
\end{array}
$$

We are now ready to present Theorem 4.6, The Fundamental Theorem of Semantics for the coercive typed $\lambda$-calculus.
Theorem 4.6: (1) The function

\[ \text{Semf} \in |L| \rightarrow \text{Ob A} + \text{Ar} |K| \]

is the unique function satisfying conditions (Semf0), (Semf1), (Semf2), (Semf3) and (Semf4).

(2) Furthermore, for each \( l \in |L| \), Semf \( l \) is a natural transformation:

\[ \text{Semf} \ l \in \text{Env} \xrightarrow{A \Rightarrow |K|} \text{Mng} \circ (\text{Type} \ l) . \]

Proof: (1) First we will verify that Semf satisfies the conditions. Let \( l \in L \), \( \alpha \in A \), \( g \in G \), \( m \in L \), \( x \in \text{Id} \), \( \tau \in \tau \).

From Corollary 4.5 and the definition of Semf,

\[ \text{Sf} \ l = \langle \text{Type} \ l, \text{Semf} \ l \rangle . \]

Since \( \text{Sf} \ l \in |V| \), we see

\[ \text{Semf} \ l \alpha \in \text{Env} \xrightarrow{\alpha \Rightarrow |K|} \text{Mng} (\text{Type} \ l \alpha) . \]

This is (Semf0). Observe

\[ \langle \text{Type} \ g, \text{Semf} \ g \rangle = \text{Sf} \ g = \text{sf} \ g = \langle \text{Type} \ g, \text{semf}' \ g \rangle . \]

Thus \( \text{Semf} \ g = \text{semf}' \ g \). This is (Semf1). From the equation

\[ \langle \text{Type} [x], \text{Semf} \ [x] \rangle = \text{Sf} \ [x] = V_x = \langle E_x, F_x \rangle \]

we get \( \text{Semf} \ [x] \alpha = F_x \alpha \), which is just (Semf2).

The equation
\langle \text{Type } [\ell \ell], \text{Semf } [\ell \ell] \rangle = \text{Sf } [\ell \ell] \\
= \forall_{ap} \langle (\text{Sf } \ell) (\text{Sf } m) \rangle \\
= \forall_{ap} \langle \langle \text{Type } \ell, \text{Semf } \ell \rangle, \\
\langle \text{Type } m, \text{Semf } m \rangle \rangle \\
= \langle T_{ap}, F_{ap} \rangle

implies

\text{Semf } [\ell \ell] \alpha = F_{ap} \alpha,

and this is (Semf3). Finally,

\langle \text{Type } [\lambda x : \tau . \ell], \text{Semf } [\lambda x : \tau . \ell] \rangle = \text{Sf } [\lambda x : \tau . \ell] \\
= \forall_{\lambda x : \tau} (\text{Sf } \ell) \\
= \forall_{\lambda x : \tau} \langle E_{\lambda x : \tau}, F_{\lambda x : \tau} \rangle

gives

\text{Semf } [\lambda x : \tau . \ell] \alpha = F_{\lambda x : \tau} \alpha,

this is (Semf4). So Semf satisfies the five conditions.

Suppose \( S : \mathcal{L} \rightarrow \mathcal{O} \mathcal{M} \mathcal{A} \mathcal{R} \mathcal{K} \mathcal{N} \) satisfies the five conditions. By condition (Semf0), for each \( \ell \in \mathcal{L}, \alpha \in \mathcal{A} \),

\[ S \ell \alpha \in \text{Env } \alpha \rightarrow \mathcal{M} \mathcal{N} \mathcal{G} (\text{Type } \ell \alpha) , \]

so that we can define

\[ S' \in \mathcal{L} \rightarrow |\Psi| \]

by

\[ S' \ell = (\text{Type } \ell, S \ell) . \]

Using (Semf2), (Semf3), and (Semf4) it is straightforward to
show that $S'$ is a homomorphism:

$$S' \in L \xrightarrow{A \rightarrow l_{g}} V.$$ 

By condition (Semfl), the restriction of $S'$ to $G$ is sf. Hence $S' = Sf$, and $S = \Pr_2 \circ S' = \Pr_2 \circ Sf = \text{Semf}$. This proves the uniqueness of Semf.

(2) We claim that for each $g \in G$, $\text{semf'} g$ is a natural transformation:

$$\text{semf'} g \in \text{Env} \xrightarrow{A \rightarrow [K]} \text{Mng} \circ (\text{Type } g).$$

This is a consequence of the obvious commutativity of the interior triangles in

Thus the image of $sf$ is contained in the subalgebra $U$ of $V$. Consequently, the image of $Sf$ is contained in $U$. Therefore, for each $\lambda$ the second component of $Sf \lambda$ is a natural transformation, i.e., Semf $\lambda$ is a natural transformation. □

The next proposition explores the semantic effect of expanding the domain of a type assignment by adding an identifier which is not free in a phrase $\lambda$. Note the role played by the naturality of Semf $\lambda$ in the proof.
Proposition 4.7: Let \( \ell \in L, \alpha \in A, \tau \in T, x \in \text{Id.} \)

Suppose \( x \notin \text{Free } \ell \). Then the diagram

\[
\begin{array}{ccc}
\text{(Env } \alpha) \times (\text{Mng } \tau) & \overset{(\text{Semf } \ell \alpha) \times 1}{\longrightarrow} & (\text{Mng } (\text{Type } \ell \alpha)) \times (\text{Mng } \tau) \\
\downarrow \pi_\alpha & & \downarrow \mu \\
\text{P}_x(\alpha, \tau) & \overset{\text{Env } \alpha}{\longrightarrow} & \text{Semf } \ell \alpha \\
\downarrow \text{Env}[\alpha | x : \tau] & & \downarrow \mu \\
\text{Mng } (\text{Type } \ell \alpha)
\end{array}
\]

commutes, where \( \mu \) is the projection onto the first component.

(Remark: The diagram makes sense by Corollary 3.11 which insures that \( \text{Mng } (\text{Type } \ell [\alpha | x : \tau]) = \text{Mng } (\text{Type } \ell \alpha) \)).

Proof: The upper half of the diagram obviously commutes, so we may turn our attention to the lower half. Denote the restriction of \( \alpha \) to \( (\text{dom } \alpha) - \{x\} \) by \( \beta \). Then \( \alpha \leq \beta \) and \( [\alpha | x : \tau] \leq \beta \). It is routine to check that

\[
\begin{array}{ccc}
\text{(Env } \alpha) \times (\text{Mng } \tau) & \overset{\pi_\alpha}{\longrightarrow} & \text{Env } \alpha \\
\downarrow \text{Env}[\alpha | x : \tau] & \overset{\text{Env } ([\alpha | x : \tau] \leq \beta)}{\longrightarrow} & \text{Env } \beta \\
\text{P}_x(\alpha, \tau) & & \text{Env } (\alpha \leq \beta)
\end{array}
\]

commutes. Since \( \alpha = [\beta | x : \alpha x] \), Corollary 3.11 gives

\( \text{Type } \ell \alpha = \text{Type } \ell \beta \).

From the naturality of \( \text{Semf } \ell \) we now get the commutativity of
Thus, the lower half of the diagram also commutes. □

In Chapter 3 we introduced Type*, a typing function applicable to phrase assignments; Type* was derived from Type, the typing function applicable to phrases. Similarly we will introduce Semf*, a semantic function that is applicable to phrase assignments and is derived from Semf.

Let I be a finite set. It is well-known and easy to check that the distinguished product functor $\Pi_{I} |K|$ for the Cartesian closed category $|K|$ gives rise in a natural way to a distinguished product functor $\Pi_{I}^{A=|K|}$ for the category $A \Rightarrow |K|$; simply construct products in $A \Rightarrow |K|$ at the level of $|K|$. To elaborate, let

$$F : I \rightarrow (A \Rightarrow |K|)$$

be an I-indexed collection of functors. Regard I as a discretely ordered poset, so that I may be viewed as a category. Then there is no difference between the functor category $I \Rightarrow |K|$ and the I-ary Cartesian product $|K|^{I}$. Let the functor
\[
F' : A \rightarrow (I \rightarrow |K|)
\]

be obtained from F by interchange of arguments:

\[
F' \alpha x = F \times \alpha.
\]

The product of F,

\[
\prod_{\alpha \in A} |K| F \in A \rightarrow |K|
\]

is defined as follows:

1. For \(\alpha \in A\),

\[
\prod_{\alpha \in A} |K| F \alpha = \prod_{\alpha \in A} |K| (F\alpha)
\]

and

2. For \(\alpha \leq A \beta\), \(\prod_{\alpha \in A} |K| F (\alpha \leq \beta)\) is the unique arrow in \(|K|\) such that for all \(x \in I\)

\[
\begin{array}{c}
\prod_{\alpha \in A} |K| (F\alpha) \xrightarrow{\text{proj}} F' \alpha x = F \times \alpha \\
\prod_{\alpha \in A} |K| F(\alpha \leq \beta) \xrightarrow{\text{proj}} F' (\alpha \leq \beta) x = F \times (\alpha \leq \beta)
\end{array}
\]

commutes.

Let \(f\) be a phrase assignment. We have a particular interest in the \((\text{dom } f)\)-indexed collection of functors

\[
V f \in \text{dom } f \rightarrow (A \rightarrow |K|)
\]

given by

\[
V f x = \text{Mng} \circ (\text{Type } (fx)).
\]
Let

\[(V \ f)' \in A \rightarrow (\text{dom } f = |K|)\]

be the functor obtained by interchanging the arguments of \(V \ f\).

**Proposition 4.8:** Let \(f \in |L|\). Then

\[\text{Env} \circ (\text{Type}^* \ f) = \Pi_{\text{dom } f}^{|A^=| \ K} (V \ f).\]

**Proof:** Let \(\alpha\) be a type assignment. If \(x \in \text{dom } f\), then

\[(V \ f)' \alpha \ x = V \ f \ x \ \alpha = \text{Mng} (\text{Type} (fx) \ \alpha) = \text{Mng} (\text{Type}^* \ f \ \alpha \ x).

Hence \((V \ f)' \alpha = \text{Mng} \circ (\text{Type}^* \ f \ \alpha)\). By the definition of \text{Env},

\[(\text{Env} \circ (\text{Type}^* \ f)) \alpha = \text{Env} (\text{Type}^* \ f \ \alpha) = \Pi_{\text{dom } f}^{|K|} (\text{Mng} (\text{Type}^* \ f \ \alpha)) = \Pi_{\text{dom } f}^{|K|} ((V \ f)' \alpha) = \Pi_{\text{dom } f}^{|A^=| \ K} (V : f) \ \alpha.

So the two functors are equal on objects.

Suppose \(\alpha \leq^A \beta\). By the monotonicity of \text{Type}^* \ f,

\[(\text{Env} \circ (\text{Type}^* \ f)) (\alpha \leq \beta) = \text{Env} ((\text{Type}^* \ f \ \alpha) \leq^A (\text{Type}^* \ f \ \beta)).\]

From the definition of \text{Env} we obtain for each \(x \in \text{dom } f\) a commutative diagram
\[ \Pi_{\text{dom} f}^{K} (\text{Mng} \circ (\text{Type}^{*} f \alpha)) = \Pi_{\text{dom} f}^{K} ((\mathbf{V}f)^{\prime} \alpha) \xrightarrow{\text{proj}} (\mathbf{V}f)^{\prime} \alpha x \]

\[ \xrightarrow{(\text{Env} \circ (\text{Type}^{*} f))(\alpha \leq \beta) \quad (\mathbf{V}f)^{\prime} (\alpha \leq \beta) x} \]

\[ \Pi_{\text{dom} f}^{K} (\text{Mng} \circ (\text{Type}^{*} f \beta)) = \Pi_{\text{dom} f}^{K} ((\mathbf{V}f)^{\prime} \beta) \xrightarrow{\text{proj}} (\mathbf{V}f)^{\prime} \beta x \]

Therefore

\[ (\text{Env} \circ (\text{Type}^{*} f))(\alpha \leq \beta) = \Pi_{\text{dom} f}^{A \Rightarrow K} (\mathbf{V}f)(\alpha \leq \beta), \]

which shows that the functors are equal on arrows. \( \square \)

For each phrase assignment \( f \in |L|^{*} \), let

\[ \text{Semf}^{*} f \in \text{Env} \xrightarrow{A \Rightarrow K} \text{Env} \circ (\text{Type}^{*} f) \]

be the unique natural transformation such that for all \( x \in \text{dom} f \),

\[ \text{Env} \circ (\text{Type}^{*} f) = \Pi_{\text{dom} f}^{A \Rightarrow K} (\mathbf{V}f) \xrightarrow{\text{proj}} \mathbf{V}f x = \text{Mng} \circ (\text{Type}(fx)) \]

commutes.

We next present the basic theorem on the semantics of substitution. Beyond its intrinsic interest, it is a most useful tool for the investigation of the semantics of \( \alpha-, \beta-, \) and \( \eta- \) reduction.
In the statement of Theorem 4.9, Godement multiplication, denoted by an infixed *, makes its first appearance. Godement multiplication is nothing more than a way of combining appropriate functors with natural transformations to get new natural transformations. Thus, let \( A, B, C, D \) be categories. Suppose

\[
F, F' \in A \to B, \ G \in B \to C, \ H, H' \in C \to D
\]

and

\[
\eta \in F, F' \to F', \quad \delta \in H, H' \to H'.
\]

We obtain natural transformations

\[
G * \eta \in G \circ F \to G \circ F', \\
\delta * G \in H \circ G \to H' \circ G.
\]

by letting

\[
(G * \eta)(A) = G(\eta A) \quad \text{for} \quad A \in \text{Ob} \ A, \\
(\delta * G)(B) = \delta(GB) \quad \text{for} \quad B \in \text{Ob} \ B.
\]

For more details about Godement multiplication, see Manes [5].

**Theorem 4.9:** Let \( \ell \in L \). For all \( f \in L^* \), the diagram

\[
\begin{array}{ccc}
\text{Env} & \xrightarrow{\text{Semp}(\text{Sub } f \ \ell)} & \text{Mng} \circ (\text{Type } (\text{Sub } f \ \ell)) \\
\downarrow \text{Semp}^f & & \downarrow \text{Mng}^*(\text{Type}(\text{Sub } f \ \ell)) \leq (\text{Type } \ell) \circ (\text{Type}^* f) \\
\text{Env} \circ (\text{Type}^* f) \circ (\text{Semp} \ \ell) \circ (\text{Type}^* f) & = & \text{Mng} \circ (\text{Type } \ell) \circ (\text{Type}^* f)
\end{array}
\]
commutes in \( A = |K| \).

**Proof:** We will show by induction on the structure of \( \ell \in L \) that for all \( f \in |L|^{\ast} \) and all \( \alpha \in A \),

\[
\begin{align*}
\text{Env} \; \alpha & \xrightarrow{\text{Semf} \; (\text{Sub} \; f \; \ell) \; \alpha} \text{Mng} \; (\text{Type} \; (\text{Sub} \; f \; \ell) \; \alpha) \\
\text{Mng} \; ((\text{Type} \; (\text{Sub} \; f \; \ell) \; \alpha) \\
& \leq (\text{Type} \; \ell \; (\text{Type}^{\ast} \; f \; \alpha)))
\end{align*}
\]

commutes in \( |K| \).

(1) Let \( \ell = g \) where \( g \in G \). By (Sub1), \( \text{Sub} \; f \; g = g \).

By (Typel),

\[
\text{Type} \; g \; \alpha = \text{type} \; g = \text{Type} \; g \; (\text{Type}^{\ast} \; f \; \alpha).
\]

Using (Semfl) and the fact that \( \text{Env} \; \text{emp}_{\ell} \) is terminal, we get commutativity of:

\[
\begin{align*}
\text{Env} \; \alpha & \xrightarrow{\text{Semf} \; g \; \alpha} \text{Mng} \; (\text{type} \; g) \\
\text{Mng} \; (\text{type} \; g)
\end{align*}
\]

as desired.
(2) Let \( \lambda = [x] \) where \( x \in \text{Id} \). On the one hand, suppose \( x \notin \text{dom } f \). Since \( \text{dom } (\text{Type}^* f \alpha) = \text{dom } f \), we are able to use (Type2) to get

\[
\text{Type } \lambda \ (\text{Type}^* f \alpha) = \text{ns}
\]

Therefore the object in the lower right corner of the square whose commutativity is desired turns out to be terminal. Thus commutativity is automatic in this case. On the other hand, suppose \( x \in \text{dom } f \). By (Sub2), \( \text{Sub } f [x] = f x \). By Theorem 3.12, the right-hand arrow is an identity morphism, and the diagram whose commutativity we want may be written as

\[
\begin{array}{ccc}
\text{Env } \alpha & \xrightarrow{\text{Semf } (fx) \alpha} & \text{Mng } (\text{Type } (fx) \alpha) \\
\downarrow \text{Semf}^* f \alpha & & \downarrow \\
\text{Env } (\text{Type}^* f \alpha) & \xrightarrow{\text{Semf } [x] (\text{Type}^* f \alpha)} & \text{Mng } (\text{Type } (fx) \alpha)
\end{array}
\]

By (Semf2), the bottom arrow is the projection on the \( x \)th component. This diagram commutes by the definition of \( \text{Semf}^* f \).

(3) Let \( \lambda = [mn] \) where \( m, n \in L \). First consider the case in which

\[
\text{Type } m \ (\text{Type}^* f \alpha) = \tau' \Rightarrow \theta' \quad \text{and} \quad \text{Type } n \ (\text{Type}^* f \alpha) = \overline{\tau'} \leq \tau'
\]

Then

\[
\text{Type } [mn] \ (\text{Type}^* f \alpha) = \theta' .
\]
By Theorem 3.12 and Theorem 2.2,

\[ \text{Type (Sub } f \ m) \alpha = \tau \Rightarrow \theta \leq \tau' \Rightarrow \theta' \ , \]
\[ \text{Type (Sub } f \ n) \alpha = \overline{\tau} \leq \overline{\tau'} \]

Since \( \text{Sub } f \ [mn] = [ (\text{Sub } f \ m) (\text{Sub } f \ n)] \),
\[ \overline{\tau} \leq \overline{\tau'} \leq \tau' \leq \tau , \text{ and } \theta \leq \theta' , \] we see
\[ \text{Type (Sub } f \ [mn]) \alpha = \theta \leq \theta' \ . \]

The diagram which we wish to show commutative may now be written as

\[ \text{Env } \alpha \rightarrow \text{ Semf (Sub } f \ [mn]) \alpha \rightarrow \text{ Mng } \theta \]
\[ \text{ (1) } \]
\[ \{ \text{Semf (Sub } f \ m) \alpha, \text{ Semf (Sub } f \ n) \alpha \} \rightarrow \text{ Ap} \]

\[ \text{Mng } (\tau \Rightarrow \theta) \times \text{Mng } \overline{\tau} \rightarrow \text{1 } \text{Mng } (\overline{\tau} \leq \tau) \rightarrow \text{Mng } (\tau \Rightarrow \theta) \times \text{Mng } \tau \]
\[ \text{ (3) } \]
\[ \text{Mng } ((\tau \Rightarrow \theta) \leq (\tau' \Rightarrow \theta')) \times \text{Mng } (\overline{\tau} \leq \overline{\tau'}) \]

\[ \text{Mng } (\tau' \Rightarrow \theta') \times \text{Mng } \overline{\tau'} \rightarrow \text{1 } \text{Mng } (\overline{\tau'} \leq \tau' ) \rightarrow \text{Mng } (\tau' \Rightarrow \theta') \times \text{Mng } \tau' \]
\[ \text{ (2) } \]
\[ \{ \text{Semf } m(\text{Type }^* f \alpha), \text{ Semf } n(\text{Type }^* f \alpha) \} \rightarrow \text{ Ap} \]

\[ \text{Env } (\text{Type }^* f \alpha) \rightarrow \text{ Semf } [mn] (\text{Type }^* f \alpha) \rightarrow \text{ Mng } \theta' \ . \]

Interior diagrams (1) and (2) commute by (Semf3), diagram (3) commutes by the induction hypothesis, and commutativity
of diagram (4) follows by the argument used to show $F_{ap}$ is natural in the proof of Theorem 4.3. So the outer square commutes, as desired. The other case is that in which

$$\text{Type} \left[ m n \right] \left( \text{Type}^* f \alpha \right) = ns$$

The relevant diagram commutes in this case because the object in its lower right corner is terminal.

(4) Let $x = \left[ \lambda x : t . m \right]$ where $x \in \text{Id}$, $t \in T$, $m \in L$.

Let $X = \text{clashset} \left( f, x, m \right)$ and $z = \text{newid} X$.

First we consider the case $x \not\in X$. In this case

$$\text{Sub} f \left[ \lambda x : t . m \right] = \left[ \lambda x : t . \left( \text{Sub} f \left| x, m \left[ x \right] \right) \right]$$

and

$$\text{Type} \left( \text{Sub} f \left[ \lambda x : t . m \right] \right) \alpha = t \Rightarrow \theta,$$

where $\theta = \text{Type} \left( \text{Sub} f \left| x, m \left[ x \right] \right) \alpha \mid x : t \right]$.

Also,

$$\text{Type} \left[ \lambda x : t . m \right] \left( \text{Type}^* f \alpha \right) = t \Rightarrow \theta', \right.$$

where $\theta' = \text{Type} \left[ m \right] \left( \text{Type}^* f \alpha \mid x : t \right]$.

By Theorem 3.12, we know $(t \Rightarrow \theta) \leq (t \Rightarrow \theta')$; hence, $\theta \leq \theta'$. Our aim is to show that

$$\begin{array}{c}
\text{Env} \alpha \\
\downarrow \text{Semf} \left( \text{Sub} f \left[ \lambda x : t . m \right] \right) \alpha \\
\downarrow \text{Mng} (t \Rightarrow \theta)
\end{array}$$

$$\begin{array}{c}
\uparrow \text{Semf}^* f \alpha \\
1 \Rightarrow \text{Mng}(\theta \leq \theta')
\end{array}$$

$$\begin{array}{c}
\text{Env} \left( \text{Type}^* f \alpha \right) \\
\downarrow \text{Semf} \left[ \lambda x : t . m \right] \left( \text{Type}^* f \alpha \right) \\
\downarrow \text{Mng} (t \Rightarrow \theta')
\end{array}$$
commutes in $|K|$. Our method is to show both composites satisfy the same universal mapping property, namely, (Cart3). Consider the diagram

$$
\begin{array}{c}
(\text{Env} \alpha) \times (\text{Mng} \tau) \\
\downarrow \downarrow \\
\text{Mng}(\tau \Rightarrow \theta') \times \text{Mng} \tau \\
\downarrow \\
\text{Mng}(\tau \Rightarrow \theta) \times \text{Mng} \tau \\
\downarrow \\
\text{Mng}(\theta \leq \theta') \times \text{Mng} \tau \\
\downarrow \\
\text{Ap}(\text{Mng} \tau, \text{Mng} \theta')
\end{array}
$$

The interior diagrams commute for the following reasons:

1. $\times$ is a functor,
2. naturality of $\text{Ap}$ in its second argument,
3. induction hypothesis,
4. (Semf4).

Thus, the outer square commutes.
Let
\[ \beta_1 = [\text{Type}^* f \alpha \mid x : \tau] \]

and
\[ \beta_2 = \text{Type}^* [f \mid x : [x]] [\alpha \mid x : \tau] \]

We claim that \( \beta_1 \) and \( \beta_2 \) agree on Free \( m \). If \( x \in \text{Free} \ m \), then
\[ \beta_1 x = \tau = \text{Type} [x] [\alpha \mid x : \tau] = \beta_2 x. \]

If \( y \in \text{Free} \ m \) and \( y \neq x \), then
\[ \beta_1 y = \text{Type} (fy) \alpha \]
\[ = \text{Type} (fy) [\alpha \mid x : \tau], \]
by observing \( x \notin X \) (hence, \( x \notin \text{Free} \ (fy) \)) and applying Cor. 3.10,
\[ = \beta_2 y. \]

Define the type assignment
\[ \beta \in ((\text{dom} f) \cup \{x\}) \cap (\text{Free} m) \to \text{Ob} \mid T \]

by
\[ \beta y = \beta_1 y = \beta_2 y, \quad y \in \text{dom} \ \beta. \]

Clearly \( \beta_1 \preceq_A \beta \) and \( \beta_2 \preceq_A \beta \).

We claim that the diagram
\[
\begin{array}{cccc}
(\text{Env} \alpha) \times (\text{Mng} \tau) & \xrightarrow{(\text{Semf}^* f \alpha) \times 1} & (\text{Env} (\text{Type}^* f \alpha)) \times (\text{Mng} \tau) \\
\downarrow \text{P} (\alpha, \tau) & \xrightarrow{\text{P} (\text{Type}^* f \alpha, \tau)} & \downarrow \text{P} (\beta_1) \\
\text{Env} [\alpha \mid x : \tau] & \xrightarrow{\text{Semf}^* [f \mid x : [x]] [\alpha \mid x : \tau]} & \text{Env} (\beta_1, \leq_\beta) \\
\downarrow \text{Env} \beta_2 & \xrightarrow{\text{Env} (\beta_2 \preceq \beta)} & \downarrow \text{Env} \beta
\end{array}
\]
commutes. The object in the lower right-hand corner is a product:

$$\operatorname{Env} \beta = \prod_{\operatorname{dom} \beta} |K| (\operatorname{Mng} \circ \beta).$$

Thus, the claim will be proved if we can verify for all $y \in \operatorname{dom} \beta$ the commutativity of the outer square of the diagram
In this last diagram the arrow $q_y$ is the projection onto the first component if $y \neq x$ and the projection onto the second component if $y = x$. Consider each interior diagram in turn; they all commute for reasons we will now explain:

(5) definition of the functor Env,

(6) definition of Semf$^*$,

(7) when $y \neq x$, $x \notin \text{Free (fy)}$ follows from $x \notin X$, and therefore commutativity follows from Proposition 3.7; when $y = x$, commutativity is a consequence of the fact that Semf $\llbracket x \rrbracket [\alpha \mid x : \tau]$ is the projection onto the $x$th component of Env$[\alpha \mid x : \tau],$

(8) definition of Semf$^*$,

(9) definition of $p_x$, Type$^* f \alpha, \tau^y$,

(10) definition of $p_x (\text{Type}^* f \alpha, \tau),$

(11) definition of the functor Env.

Thus, the entire diagram commutes, proving the claim.

Next consider the diagram
Commutativity of interior diagram (12) has been demonstrated. Diagram (13) commutes because \( \times \) is a functor. Diagram (14) commutes by (Semf4). The diagrams labelled (15) and (16) are commutative since \( \text{Semf} \ m \) is a natural transformation. Thus the outer square commutes.

Compare the outer squares of diagrams \((++)\) and \((+++)\). By applying the uniqueness assertion of (Cart3) we get commutativity of \((+)\).
Finally, consider the case \( x \in X \). In the case
\[
\text{Sub } f \left[ \lambda x : \tau . m \right] = \left[ \lambda z . \left( \text{Sub } [f \mid x : [z]] \ m \right) \right],
\]
and this time
\[
\text{Type } \left( \text{Sub } f \left[ \lambda x : \tau . m \right] \right) = \tau \Rightarrow \theta,
\]
where \( \theta = \text{Type } \left( \text{Sub } [f \mid x : [z]] \ m \right) [\alpha \mid z : \tau] \).
Again,
\[
\text{Type } \left[ \lambda x : \tau . m \right] (\text{Type}^* \ f \ \alpha) = \tau \Rightarrow \theta',
\]
where \( \theta' = \text{Type } m [\text{Type}^* \ f \ \alpha \mid x : \tau] \). By slightly altering the argument in the preceding case, we again obtain commutativity of \((\dagger)\). □

The next three theorems study in turn the semantics of \( \alpha \)-reduction, \( \beta \)-reduction, and \( \eta \)-reduction.

**Theorem 4.10:** Let \( x, y \in \text{Id} \), \( \lambda \in L \), \( \tau \in T \).
Suppose \( y \notin \text{Free } \lambda \) and \( m = \text{Sub } \left[ \text{emp}_\lambda \mid x : [y] \right] \lambda \).
Then
\[
\text{Sem}_f \left[ \lambda x : \tau . \lambda \right] = \text{Sem}_f \left[ \lambda y : \tau . m \right].
\]

**Proof:** Let \( \alpha \in A \). By \( \hat{\alpha} \in A \) we mean the restriction of \( \alpha \) to \( \text{dom } \alpha - \{ y \} \). Thus \( \hat{\alpha} \in \text{dom } \alpha - \{ y \} \rightarrow \text{Ob } |\tau| \). Extend the phrase assignment \( \left[ \text{emp}_\lambda \mid x : [y] \right] \) to
\[
f \in (\text{dom } \hat{\alpha}) \quad \{ x \} \rightarrow |L|
\]
via
\[
\begin{align*}
[z] & \quad \text{if } z \neq x \\
f z & \\
[y] & \quad \text{if } z = x.
\end{align*}
\]
By Theorem 3.3, \( \text{Sub } f = \text{Sub } \{ \text{emp}_L \mid x : \tau \} \), so that \( m = \text{Sub } f \ell \). It is easy to see that

\[
\text{Type}^* f [\alpha \mid y : \tau] = [\hat{\alpha} \mid x : \tau].
\]

Therefore

\[
\text{Semf}^* f [\alpha \mid y : \tau] \in \text{Env} [\alpha \mid y : \tau] \xrightarrow{\sim} \text{Env} [\hat{\alpha} \mid x : \tau].
\]

Using the universal mapping property that defines \( P_x(\alpha, \tau) \), it is an easy exercise to prove that

\[
\begin{array}{c}
\text{Env} \alpha \times \text{Mng} \tau \\
\downarrow \text{iso} \\
\text{Env} [\alpha \mid y : \tau]
\end{array}
\xrightarrow{\sim}
\begin{array}{c}
P_x(\hat{\alpha}, \tau) \\
\downarrow \text{Semf}^* f [\alpha \mid y : \tau]
\end{array}
\xrightarrow{\sim}
\begin{array}{c}
\text{Env} [\hat{\alpha} \mid x : \tau]
\end{array}
\]

commutes, where iso is the obvious isomorphism. By Proposition 3.13, we may let

\[
\theta = \text{Type } \ell [\alpha \mid x : \tau] = \text{Type } m [\alpha \mid y : \tau];
\]

then also \( \theta = \text{Type } \ell [\hat{\alpha} \mid x : \tau] \) (by Corollary 3.11) and

\[
\tau \Rightarrow \theta = \text{Type } [\lambda x : \tau. \ell] \alpha = \text{Type } [\lambda y : \tau. m] \alpha.
\]

Consider the diagram
The interior diagrams commute for the following reasons:

1. naturality of $P_x'$,
2. definition of $P_y(\alpha, \tau)$,
3. (Semf 4),
4. Theorem 4.9,
5. naturality of $\text{Semf } \ell$,
6. by earlier remarks.

Hence the outer square commutes. Apply (Semf 4) again to get $\text{Semf } \left[ \lambda x : \tau. \ell \right] \alpha = \text{Semf } \left[ \lambda y : \tau. m \right] \alpha$. □
Theorem 4.11: Let $x \in \text{Id}, \tau \in T, \lambda, m \in L$.
Suppose $n = \text{Sub} [\text{emp}_L | x : m] \lambda$ and $n' = \| (\lambda x : \tau. \lambda)m \|$.
Then

$\text{Semf } n \xrightarrow{} \text{Mng } \circ \text{ (Type } n \text{)}$

$\text{Env } \xrightarrow{} \text{Mng } \ast \text{ (Type } n \leq \text{ Type } n' \text{)}$

$\text{Semf } n' \xrightarrow{} \text{Mng } \circ \text{ (Type } n' \text{)}$

commutes in $A = |K|$.

(Remark: Thus, if Type $n = \text{Type } n'$, it follows that Semf $n = \text{Semf } n'$.)

Proof: Note that Type $n \leq \text{Type } n'$ by Proposition 3.14. Let $\alpha \in A, \theta = \text{Type } n \alpha, \theta' = \text{Type } n' \alpha$. We must verify that

$\text{Semf } n \alpha \xrightarrow{} \text{Mng } \theta$

$\text{Env } \alpha \xrightarrow{} \text{Mng } (\theta \leq \theta')$

$\text{Semf } n' \alpha \xrightarrow{} \text{Mng } \theta'$

commutes. If $\theta' = ns$, then this diagram obviously commutes. So suppose that $n' \neq ns$. Upon recalling that $n' = \| (\lambda x : \tau. \lambda)m \|$, we see this can only happen if

$\text{Type } m \alpha = \overline{\tau},$ where $\overline{\tau} \leq \tau$. 
As in the proof of Proposition 3.14, the phrase assignment

\[ f_\alpha \in (\text{Free } \lambda) \cup (\text{dom } \alpha) \cup \{x\} \rightarrow |L| \]

is given by

\[ f_\alpha y = \begin{cases} \llbracket y \rrbracket, & \text{if } y \neq x \\ m, & \text{if } y = x \end{cases} \]

Recall that

\[ \text{Type } n = (\text{Type } \lambda) \circ (\text{Type }^* f_\alpha), \]

and

\[ \text{Type }^* f_\alpha \alpha \leq [\alpha | x : \tau]. \]

This implies that

\[ \emptyset = \text{Type } n \alpha = \text{Type } \lambda (\text{Type }^* f_\alpha \alpha), \]

so that

\[ \text{Semf } \lambda (\text{Type }^* f_\alpha \alpha) \in \text{Env } (\text{Type }^* f_\alpha \alpha) \xrightarrow{|K|} \text{Mng } \emptyset. \]

For each \( y \in \text{dom } [\alpha | x : \tau] \) consider the diagram
The interior diagrams commute for the following reasons:

1. Definition of Env,
2. Definition of Semf^*
3. Definition of p_{x,\alpha,\bar{\tau}}^, together with observation that Semf(f_{\alpha,y})\alpha = proj y when y \neq x, and Semf(f_{\alpha,y'})\alpha = Semf m \alpha when y = x,
4. Definition of P_{x}(\alpha, \bar{\tau})

By the universal mapping property for products, we see that

\[
\begin{array}{ccc}
\text{Env }\alpha & \xrightarrow{(l, \text{Semf } m \alpha)} & (\text{Env }\alpha) \times (\text{Mng}\bar{\tau}) \\
\text{Semf}^* f_{\alpha} \alpha & \downarrow & P_{x}(\alpha, \bar{\tau}) \\
\text{Env}(\text{Type}^* f_{\alpha} \alpha) & \xrightarrow{\text{Env}(\text{Type}^* f_{\alpha} \alpha \equiv [\alpha | x: \bar{\tau}])} & \text{Env}[\alpha | x: \bar{\tau}]
\end{array}
\]

commutes.
Finally, consider the diagram:

\[
\text{Semf} \varphi \alpha \quad \text{Env} \alpha \quad \text{Semf} \varphi' \alpha \quad \text{Mng} \theta' \quad \text{Mng} \theta
\]

(6)

\[
\text{Semf}^* f \alpha \quad \text{Env} \alpha \times \text{Mng} \overline{\tau} \quad \text{Mng}(\tau \Rightarrow \theta') \times \text{Mng} \overline{\tau}
\]

(5)

\[
\text{P}_x(\alpha, \overline{\tau}) \quad \text{Env}[\alpha | x : \overline{\tau}] \quad \text{Enva} \times \text{Mng} \tau
\]

(9)

(11)

(12)

(8)

\[
\text{Env}(\ldots \ldots) \quad \text{Env}(\ldots \ldots)
\]

\[
\text{Env}(\text{Type} \! f \alpha) \quad \text{Env}(\text{Type} \! f' \alpha)
\]

(7)

\[
\text{Semf} \lambda(\text{Type} \! f \alpha) \quad \text{Semf} \lambda(\alpha | x : \overline{\tau})
\]

\[
\text{Mng} \theta \quad \text{Mng} \overline{\tau} \quad \text{Mng} \theta'
\]
We claim this diagram commutes. This follows from the commutativity of each of the numbered subdiagrams, which commute for the following reasons:

5. commutativity demonstrated earlier in this proof,
6. Theorem 4.9,
7. naturality of Semf \( \ell \),
8. functoriality of Env,
9. naturality of \( P_x \),
10. (Semf4),
11. clearly commutes,
12. (Semf3).

Theorem 4.12: Let \( x \in \text{Id}, \tau \in \mathcal{T}, \) and \( \ell \in \cal L \). Suppose \( m = [\lambda x:\tau.[\ell[[x]]]], \) \( x \notin \text{Free } \ell, \) and

\[
\text{Type } \ell \leq A \Rightarrow |\mathcal{T}| \quad \text{Type } m.
\]

Then

\[
\begin{array}{ccc}
\text{Semf } \ell & \xrightarrow{\text{Mng}} & \text{Type } \ell \\
\text{Env} & & \text{Mng } \ast (\text{Type } \ell \leq \text{Type } m) \\
\text{Semf } m & \xrightarrow{\text{Mng}} & \text{Type } m \\
\end{array}
\]

commutes in \( A \Rightarrow |K| \).

(Remark: As in Theorem 4.11, we conclude that \( \text{Semf } \ell = \text{Semf } m \) if \( \text{Type } \ell = \text{Type } m \).)
Proof: Let $\alpha \in A$. We can write

$$\text{Type } m \alpha = \tau \Rightarrow \theta$$

where $\theta = \text{Type } [l [x]] [\alpha \mid x: \tau]$. Since

Type $l \alpha \leq \text{Type } m \alpha$, we can write

$$\text{Type } l \alpha = \tau' \Rightarrow \theta'$$

where $\tau \leq \tau'$ and $\theta' \leq \theta$. Since $x \notin \text{Free } l$, Corollary 3.11 tells us

$$\text{Type } l [\alpha \mid x: \tau] = \tau' \Rightarrow \theta'$$

Combining this with $\tau \leq \tau'$ and

$$\text{Type } [x] [\alpha \mid x: \tau] = [\alpha \mid x: \tau] x = \tau$$

we see

$$\theta = \text{Type } [l [x]] [\alpha \mid x: \tau] = \theta'$$

Let

$$\mu = \text{Mng } (\tau \leq \tau') \in \text{Mng } \frac{\tau}{K} \rightarrow \text{Mng } \tau'$$

We must verify that

\[
\begin{array}{ccc}
\text{Semf } l \alpha & \rightarrow & \text{Mng } (\tau' \Rightarrow \theta) \\
\text{Env } \alpha & \rightarrow & \mu = 1 \\
\text{Semf } m \alpha & \rightarrow & \text{Mng } (\tau \Rightarrow \theta)
\end{array}
\]
commutes in \(|K|\). We know that \(\text{Semf}_m \alpha\) is the unique arrow in \(|K|\) such that

\[
\begin{array}{c}
\text{Env} \alpha \times (\text{Mng}_\tau) \xrightarrow{(\text{Semf}_m \alpha) \times 1} (\text{Mng}(\tau \Rightarrow \theta)) \times (\text{Mng}_\tau) \\
\xleftarrow{P_x(\alpha, \tau)} \quad \xrightarrow{\text{Ap}(\text{Mng}_\tau, \text{Mng}_\theta)} \\
\text{Env} \left[ \alpha \mid x : \tau \right] \text{Semf} \llbracket \lambda \ [x] \rrbracket \llbracket \alpha \mid x : \tau \rrbracket \xrightarrow{Mng \ \theta}
\end{array}
\]

commutes. Thus, if we can verify the commutativity of

\[
\begin{array}{c}
\text{Env} \alpha \times (\text{Mng}_\tau) \xrightarrow{((\mu \Rightarrow 1) \circ (\text{Semf} \ & \alpha)) \times 1} (\text{Mng}(\tau \Rightarrow \theta)) \times (\text{Mng}_\tau) \\
\xleftarrow{(\text{Semf} \ & \alpha) \times 1} \quad \xrightarrow{(\mu \Rightarrow 1) \times 1} \\
\text{Env} \left[ \alpha \mid x : \tau \right] \text{Semf} \llbracket \lambda \ [x] \rrbracket \llbracket \alpha \mid x : \tau \rrbracket \xrightarrow{Mng \ \theta}
\end{array}
\]

\[
\begin{array}{c}
\text{Env} \alpha \times (\text{Mng}_\tau) \xrightarrow{((\mu \Rightarrow 1) \circ (\text{Semf} \ & \alpha)) \times 1} (\text{Mng}(\tau \Rightarrow \theta)) \times (\text{Mng}_\tau) \\
\xleftarrow{(\text{Semf} \ & \alpha) \times 1} \quad \xrightarrow{(\mu \Rightarrow 1) \times 1} \\
\text{Env} \left[ \alpha \mid x : \tau \right] \text{Semf} \llbracket \lambda \ [x] \rrbracket \llbracket \alpha \mid x : \tau \rrbracket \xrightarrow{Mng \ \theta}
\end{array}
\]

\[
\begin{array}{c}
\text{Env} \alpha \times (\text{Mng}_\tau) \xrightarrow{((\mu \Rightarrow 1) \circ (\text{Semf} \ & \alpha)) \times 1} (\text{Mng}(\tau \Rightarrow \theta)) \times (\text{Mng}_\tau) \\
\xleftarrow{(\text{Semf} \ & \alpha) \times 1} \quad \xrightarrow{(\mu \Rightarrow 1) \times 1} \\
\text{Env} \left[ \alpha \mid x : \tau \right] \text{Semf} \llbracket \lambda \ [x] \rrbracket \llbracket \alpha \mid x : \tau \rrbracket \xrightarrow{Mng \ \theta}
\end{array}
\]

\[
\begin{array}{c}
\text{Env} \alpha \times (\text{Mng}_\tau) \xrightarrow{((\mu \Rightarrow 1) \circ (\text{Semf} \ & \alpha)) \times 1} (\text{Mng}(\tau \Rightarrow \theta)) \times (\text{Mng}_\tau) \\
\xleftarrow{(\text{Semf} \ & \alpha) \times 1} \quad \xrightarrow{(\mu \Rightarrow 1) \times 1} \\
\text{Env} \left[ \alpha \mid x : \tau \right] \text{Semf} \llbracket \lambda \ [x] \rrbracket \llbracket \alpha \mid x : \tau \rrbracket \xrightarrow{Mng \ \theta}
\end{array}
\]
then the commutativity of the triangle in question follows. The interior diagrams (1), (2), and (3) commute for the following reasons:

(1) $- \times 1$ is a functor,
(2) the definition of $\mu \Rightarrow 1$ (see Chapter I),
(3) $(\text{Semf3})$.

The proof that (4) commutes involves a short argument. The demonstration is based on the fact that $(\text{Semf} f \alpha) \times 1$ is the unique morphism such that

\[
\begin{array}{ccccccc}
\text{Env } \alpha & \xleftarrow{\pi} & (\text{Env } \alpha ) \times (\text{Mng } \tau) & \xrightarrow{p_{x, \alpha, \tau} x} & \text{Mng } \tau \\
\downarrow \text{Semf } f \alpha & & & & \downarrow \text{Semf } f \alpha \times 1 \\
\text{Mng } (\tau' \Rightarrow \delta) & \xleftarrow{\pi_1} & (\text{Mng } (\tau' \Rightarrow \delta)) \times (\text{Mng } \tau) & \xrightarrow{\pi_2} & \text{Mng } \tau
\end{array}
\]

commutes where $\pi_1$ and $\pi_2$ are the indicated projections. Consider the diagram

\[
\begin{array}{ccccccc}
\text{Env } \alpha & \xleftarrow{\pi} & (\text{Env } \alpha) \times (\text{Mng } \tau) & \xrightarrow{p_{x, \alpha, \tau} x} & \text{Mng } \tau \\
\downarrow \text{Semf } f \alpha & & & & \downarrow \text{P}_x \langle \alpha, \tau \rangle \\
\text{Env } \langle \alpha | x : \tau \rangle & \xrightarrow{\text{Semf } f \alpha [\alpha | x : \tau]} & \langle \text{Semf } f [\alpha | x : \tau], \text{Semf } f [\alpha | x : \tau] \rangle & \xrightarrow{\pi_2} & \text{Mng } \tau
\end{array}
\]

(5) (6) (7) (8)
Interior diagram (5) commutes by Theorem 4.7 and interior
diagram (6) commutes by the definition of $P_x(\alpha, \tau)$ along
with the observation that $\text{Semf} \{x\} [\alpha \mid x: \tau]$ is the
projection on the $x$th component. Finally, (7) and (8)
commute by the universal mapping property defining
$\langle \text{Semf} \& \{\alpha \mid x: \tau\}, \text{Semf} \{x\} [\alpha \mid x: \tau]\rangle$. Hence the entire
diagram commutes, and commutativity of (4) is now immediate.

CHAPTER V

CARTESIAN CLOSED CATEGORIES

In order to investigate the semantics of ALGOL-like languages it is necessary to specify a Cartesian closed category, among whose arrows are found the denotations of program fragments. In this chapter we will establish that certain semantically important categories are Cartesian closed. Because of the motivation given in Chapter I, we are particularly interested in categories whose objects are functors.

We begin by establishing some notational conventions. If \( f \) is a function and \( X \) is a set of arguments to which \( f \) may be applied, then we write

\[
fx = \{fx \mid x \in X\}.
\]

Similarly, if \( F \) is a set of functions and \( x \) is an argument in the domain of each function in \( F \), then we write

\[
Fx = \{fx \mid f \in F\}.
\]

We may even write

\[
FX = \{fx \mid f \in F, x \in X\}.
\]

A poset is a pair consisting of an underlying set and a partial order. Routinely, a poset is viewed as a category whose collection of objects is the underlying set. Thus, whenever \( P \) is a poset, its underlying set may be denoted as \( \text{Ob} \ P \).

We remarked in Chapter I that \( \text{Set} \) is a Cartesian closed category. The point of the next two theorems is that the
structure of \text{Set} as a Cartesian closed category can be extended to \( \text{Pdom} \), and that the result of that extension can be restricted to \( \text{Dom} \), so that both \( \text{Pdom} \) and \( \text{Dom} \) are Cartesian closed categories.

As a prelude to what follows note that both \( \text{Pdom} \) and \( \text{Dom} \) are categories with finite products specifically given; the products are constructed by partially ordering componentwise the products in \( \text{Set} \) of the underlying sets.

**Theorem 5.1**: The category \( \text{Pdom} \) is Cartesian closed.

**Proof**: The distinguished terminal object and the distinguished binary product functor for \( \text{Pdom} \) are constructed in the obvious way from their counterparts in \( \text{Set} \). In particular, \( \text{Pdom}_{\text{term}} \) is \( \text{E Set}_{\text{term}} \), a singleton set regarded as a predomain.

We now turn to exponentiation. Suppose \( Q \) and \( R \) are predomains; we will define the predomain \( Q \overset{\text{Pdom}}{\longrightarrow} R = Q \Rightarrow R \).

The underlying set of \( Q \Rightarrow R \) is

\[
\text{Ob} \ (Q \Rightarrow R) = Q \longrightarrow_{\text{Pdom}} R,
\]

the set of continuous functions from \( Q \) to \( R \). The partial order for \( Q \Rightarrow R \) is defined by \( f \leq_Q \Rightarrow_R f' \) if \( f q \leq_R f' q \) for all \( q \in \text{Ob} \ Q \). We must show that \( Q \Rightarrow R \) is directed-complete. To this end, let \( D \) be a directed subset of \( Q \Rightarrow R \). Define the function

\[
g_D \in \text{Ob} \ Q \longrightarrow \text{Ob} \ R
\]

by

\[
g_D q = \bigsqcup_R (Dq) \text{ for all } q \in \text{Ob} \ Q.
\]

We claim \( g_D \) is continuous. Let \( X \) be a directed subset of \( Q \).

Continuity of \( g_D \) follows from the computation
\[ g_D (\bigcup Q X) = \bigcup_R (D (\bigcup Q X)) \]
\[ = \bigcup_R (DX), \text{ since every function in } D \text{ is continuous,} \]
\[ = \bigcup_R \{ f x \mid f \in D, x \in X \} \]
\[ = \bigcup_R \{ \bigcup_R (Dx) \mid x \in X \} \]
\[ = \bigcup_R \{ g_D x \mid x \in X \} \]
\[ = \bigcup_R (g_D X). \]

Thus

\[ g_D \in Q \xrightarrow{P\text{dom}} R. \]

Next we claim that \( g_D \) is the least upper bound of \( D \). Surely, if \( f \in D \) then, for all \( q \in \text{Ob } Q \),

\[ f q \leq_R \bigcup_R (Dq) = g_D q; \]

so \( f \leq_{Q \Rightarrow R} g_D \) and \( g_D \) is an upper bound for \( D \). If \( h \) is an arbitrary upper bound for \( D \), then \( f \leq_{Q \Rightarrow R} h \) for \( f \in D \), which implies that for all \( q \in \text{Ob } Q \)

\[ g_D q = \bigcup_R (Dq) \leq_R h q. \]

So \( g_D \) is indeed the least upper bound of \( D \). This proves \( Q \Rightarrow R \)

Application in \( P\text{dom} \) is the usual functional application.

In more detail, suppose \( Q \) and \( R \) are predomains, and define

\[ \text{Ap } (Q, R) = \text{Ap } \varepsilon (Q \Rightarrow R) \times Q \xrightarrow{P\text{dom}} R \]

by

\[ \text{Ap } (f, q) = f q. \]

It is necessary to verify that \( \text{Ap} \) is continuous. Suppose \( D \)

is a directed subset of \( (Q \Rightarrow R) \times Q \). Let

\[ D_1 = \{ f \mid (f, x) \in D \text{ for some } x \in \text{Ob } Q \}, \]
\[ D_2 = \{ x \mid (f, x) \in D \text{ for some } f \in \text{Ob } (Q \Rightarrow R) \}. \]

Then \( D_1 \) is a directed subset of \( Q \Rightarrow R \), \( D_2 \) is a directed subset of \( Q \).
\[
\bigcup_{(Q \rightarrow R) \times Q} \mathcal{D} = \left( \bigcup_{Q \rightarrow R} \mathcal{D}_1, \bigcup_{Q \rightarrow R} \mathcal{D}_2 \right),
\]
and
\[
\bigcup_{R} (\text{Ap } \mathcal{D}) = \bigcup_{R} (\mathcal{D}_1 \mathcal{D}_2)
\]
The computation
\[
\text{Ap } \left( \bigcup_{(Q \rightarrow R) \times Q} \mathcal{D} \right) = \text{Ap } \left( \bigcup_{Q \rightarrow R} \mathcal{D}_1, \bigcup_{Q \rightarrow R} \mathcal{D}_2 \right)
\]
\[
= \bigcup_{Q \rightarrow R} \mathcal{D}_1 \left( \bigcup_{Q \rightarrow R} \mathcal{D}_2 \right)
\]
\[
= \bigcup_{R} (\mathcal{D}_1 \mathcal{D}_2),
\]
since each function in \( \mathcal{D}_1 \) is continuous
\[
= \bigcup_{R} (\text{Ap } \mathcal{D})
\]
shows that \( \text{Ap} \) is continuous.

Next, for predomains \( P, Q, \) and \( R \) consider the function
\[
\text{Ab } \langle P, Q, R \rangle = \text{Ab } e (P \times Q \xrightarrow{P_{dom}} R) \rightarrow (\text{Ob } P \rightarrow (\text{Ob } Q \rightarrow \text{Ob } R))
\]
defined by
\[
\text{Ab } f \ p \ q = f \langle p, q \rangle
\]
where \( f : P \times Q \xrightarrow{P_{dom}} R, \ p \in \text{Ob } P, \ q \in \text{Ob } Q. \)

We claim that
\[
\text{Ab } e (P \times Q \xrightarrow{P_{dom}} R) \rightarrow (P \xrightarrow{P_{dom}} (Q \rightarrow R))
\]
First we will show \( \text{Ab } f \ p \) is continuous, where \( f : P \times Q \xrightarrow{P_{dom}} R \)
and \( p \in \text{Ob } P. \) Let \( \mathcal{D} \) be a directed subset of \( Q. \) Then \( \{p\} \times \mathcal{D} \)
is a directed subset of \( P \times Q \) and
\[
\bigcup_{P \times Q} \{\{p\} \times \mathcal{D}\} = \{p, \bigcup_{Q} \mathcal{D}\}.
\]
Compute:
\[
\text{Ab } f \ p \ (\bigcup_{Q} \mathcal{D}) = f \langle p, \bigcup_{Q} \mathcal{D} \rangle
\]
\[
= f \left( \bigcup_{P \times Q} \{\{p\} \times \mathcal{D}\} \right)
\]
\[
= \bigcup_{R} \left( f \left( \{p\} \times \mathcal{D} \right) \right), \text{ by the continuity of } f
\]
\[
= \bigcup_{R} \left( \text{Ab } f \ p \ \mathcal{D} \right).
\]
So \( \text{Ab } f \ p \) is continuous. Second we will show \( \text{ab } f \) is continuous.

Let \( X \) be a directed subset of \( P. \) For all \( q \in \text{Ob } Q, \)
$X \times \{q\}$ is a directed subset of $P \times Q$ and

$$\bigcup_{P \times Q} (X \times \{q\}) = \bigcup_{P} X, q.$$

Now the equation

$$Ab f \ (\bigcup_{P} X) = \bigcup_{Q \in R} (Ab f \ X)$$

follows from the observation that for all $q \in Ob Q$

$$Ab f \ (\bigcup_{P} X) q = f \ (\bigcup_{P} X, q)$$

$$= f \ (\bigcup_{P \times Q} (X \times \{q\}))$$

$$= \bigcup_{R} (f \ (X \times \{q\})),$$

by the continuity of $f$ ,

$$= \bigcup_{R} (Ab f \ X q)$$

$$= \bigcup_{Q \in R} (Ab f \ X) q.$$

This proves the claim.

To finish the proof the $Pdom$ is a Cartesian closed category, for any $f \in P \times Q \xrightarrow{Pdom} R$ consider the diagram

$$\begin{array}{ccc}
P \times Q & \xrightarrow{f' \times 1_Q} & (Q \Rightarrow R) \times Q \\
\downarrow f & & \downarrow Ap \\
R & & \\
\end{array}$$

If we take $f' = Ab f$, then the diagram clearly commutes.

Furthermore, if $f'$ makes the diagram commute, then by applying the forgetful functor $U \in Pdom \rightarrow Set$ we get a commutative diagram in the Cartesian closed category $Set$, from which it follows that $f' = Ab f$. □

Suppose $K$ is a Cartesian closed category, and $M$ is a full subcategory of $K$. Suppose further that the distinguished product functors for $K$ give objects of $M$ when applied to objects of $M$, and that the terminal object $K$ is an object of $M$. What else do we need to know about $M$ to conclude that it, too, is a Cartesian closed category? The answer is simple: if, for all pairs $M, M'$ of objects of $M$, $M \xrightarrow{K} M'$ is an object of
$\mathcal{M}$, then $\mathcal{M}$ is a Cartesian closed category with $\Rightarrow$, $\text{Ap}$, and $\text{Ab}$ for $\mathcal{M}$ defined by restricting to $\mathcal{M}$ the corresponding entities for $\mathcal{K}$. We apply this observation in the proof of the following theorem.

**Theorem 5.2:** The category $\text{Dom}$ is Cartesian closed.

**Proof:** Suppose $\mathcal{P}$ and $\mathcal{Q}$ are domains. It suffices to show that the predomain $\mathcal{P} \xrightarrow{\text{Pdom}} \mathcal{Q}$ is a domain by exhibiting a minimal element. Clearly the minimal element is given by

$$1_\mathcal{P} \Rightarrow x = 1_\mathcal{Q}, \text{ for all } x \in \text{Ob } \mathcal{P}.$$  

Of course, $\text{Sdom}$ and $\text{Dom}$ share the same distinguished terminal object. Also, restriction of the distinguished product functors for $\text{Dom}$ to $\text{Sdom}$ yields distinguished product functors for $\text{Sdom}$. Perhaps surprisingly, we will show in the next theorem that $\text{Sdom}$ is not Cartesian closed. Note that this is a rather strong claim because it asserts not only that the restrictions of $\mathcal{P}_{\text{Dom}}$, $\text{Ap}_{\text{Dom}}$, and $\text{Ab}_{\text{Dom}}$ do not behave properly but also that no other way of defining $\mathcal{P}_{\text{Sdom}}$, $\text{Ap}_{\text{Sdom}}$, and $\text{Ab}_{\text{Sdom}}$ works.

As a preliminary, consider an example due to John C. Reynolds. Let $\mathcal{P}$ be the domain with Hasse diagram

```
  \text{T}_\mathcal{P}
  \bigg|          \bigg|
  \text{I}_\mathcal{P}
```

Let

$$f \in \mathcal{P} \times \mathcal{P} \xrightarrow{\mathcal{P}_{\text{Dom}}} \mathcal{P}$$

be the strict, continuous function given by
\[ f(x) = \begin{cases} \tau_P & \text{if } x \neq (\uparrow_P, \downarrow_P) \\ \downarrow_P & \text{if } x = (\uparrow_P, \downarrow_P) \end{cases} \]

Then
\[ \text{Ab}_{\text{Dom}} f \in P \xrightarrow{\text{Dom}} (P \xrightarrow{\text{Dom}} P) \]
is not strict because the computation
\[ \text{Ab}_{\text{Dom}} f \uparrow_P P = f(\uparrow_P, P) = \downarrow_P P \]
for \( p \in \text{Ob} \ P \), shows
\[ \text{Ab}_{\text{Dom}} f \downarrow_P = \downarrow_P \neq \uparrow_P \rightarrow P. \]

**Theorem 5.3:** The category Sdom is not Cartesian closed.

**Proof:** Suppose to the contrary that Sdom is Cartesian closed with internal hom functor \( \xrightarrow{\text{Sdom}} \), application \( \text{Ap}_{\text{Sdom}} \), and abstraction \( \text{Ab}_{\text{Sdom}} \).

Suppose \( Q \) and \( R \) are arbitrary domains. To avoid subscripts on subscripts, let
\[ X = Q \xrightarrow{\text{Sdom}} R \text{ and } Y = Q \xrightarrow{\text{Dom}} R. \]

Let
\[ H = \text{Ap}_{\text{Sdom}} (Q, R) \text{ and } K = \text{Ab}_{\text{Dom}} (X, Q, R), \]
so that
\[ H \in X \times Q \xrightarrow{\text{Sdom}} R \text{ and } K H \in X \xrightarrow{\text{Dom}} Y. \]

We claim that \( K H \) is strict. To see this, let \( S \) be a one-point domain, and let
\[ g \in S \times Q \xrightarrow{\text{Sdom}} R \]
be the constant function which maps everything to \( \uparrow_R \).
Then the diagram
\[ \begin{array}{ccc}
S \times Q & \xrightarrow{(\text{Ab}_{\text{Sdom}} (S, Q, R)g) \times \downarrow_Q} & X \times Q \\
g & & \downarrow H \\
& & R
\end{array} \]
commutes in $Sdom$. For $q \in \text{Ob } Q$, we compute as follows:

\[
K X q = H (1_X, q), \text{ by definition of } K,
\]

\[
= (H \circ (\text{Ab}_{Sdom} (S, Q, R) \times 1_Q)) (1_S, q)
\]

\[
= g (1_S, q)
\]

\[
= 1_R
\]

\[
= 1_Y q
\]

Hence, $K X = 1_Y$, as desired.

Let $P$ be any domain, and let

\[
f \in P \times Q \xrightarrow{\text{Sdom}} R.
\]

Consider the diagram

\[
\begin{array}{ccc}
P \times Q & \xrightarrow{(\text{Ab}_{Sdom} (P, Q, R) f) \times 1} & X \times Q \\ & \downarrow f & \downarrow H \\ & X \times Q & \xrightarrow{(K H) \times 1} Y \times Q \\ & \downarrow H & \downarrow R \\ & R & \xrightarrow{\text{Ab}_{Dom} (Q, R)}
\end{array}
\]

The left-hand triangle commutes in $Sdom$ and the right-hand triangle commutes in $Dom$. Thus the entire diagram commutes in $Dom$. Consequently

\[
\text{Ab}_{Dom} (P, Q, R) f = (K H) \circ (\text{Ab}_{Sdom} (P, Q, R) f),
\]

and we have expressed $\text{Ab}_{Dom} (P, Q, R) f$ as the composition of two strict functions, so it too must be strict. However, we have earlier seen an example of a strict continuous function $f$ and domains $P, Q, R$ (actually all are equal in the example) such that $\text{Ab}_{Dom} (P, Q, R) f$ is not strict. Therefore $Sdom$ is not Cartesian closed. □

In the rest of this chapter $\Sigma$ denotes a category, and our focus is on the functor category $K = \Sigma \to Pdom$ and its subcategories.

In Chapter IV we discussed how a product functor such as $\Pi_{\text{pdom}}$ induces a product functor $\Pi_{\text{K}}$. Furthermore each terminal object of $Pdom$ determines a terminal object in $K$,
namely the constant functor which assigns the terminal object to each object of \( \Sigma \).

The next theorem is nontrivial in the sense that it is not deducible from the well-known fact (see [3]) that \( \Sigma \to \text{Top} \) is a topos whenever \( \text{Top} \) is a topos, because \( P\text{dom} \) is not a topos. To see that \( P\text{dom} \) is not a topos simply observe that in \( P\text{dom} \) there are arrows which are both monomorphisms and epimorphisms but not isomorphisms (e.g. use a nonconstant arrow from a discrete, two-element predomain to a nondiscrete two element predomain). This sort of thing doesn't happen in a topos. Again, see [3].

**Theorem 5.4:** For any category \( \Sigma \), the functor category \( K = \Sigma \to P\text{dom} \) is Cartesian closed.

**Proof:** We have seen that distinguished products and terminal objects in \( K \) are inherited from \( P\text{dom} \). Therefore, we may turn our attention to the nature of exponentiation in \( K \).

By

\[
\text{hom}_\Sigma \in \Sigma^{\text{op}} \times \Sigma \to \text{Set}
\]

we mean the usual hom functor. Thus for objects \( S \) and \( T \) of \( \Sigma \) we have \( \text{hom}_\Sigma (S, T) = S \to T \). As before, let

\[
E \in \text{Set} \to P\text{dom}
\]

be the embedding functor which associates a discrete partial order with each set. Hence

\[
E \circ \text{hom}_\Sigma \in \Sigma^{\text{op}} \times \Sigma \to P\text{dom}
\]

Denote by

\[
\text{hom}^\Sigma \in \Sigma^{\text{op}} \to (\Sigma \to P\text{dom}) = \Sigma^{\text{op}} \to K
\]

the curried version of \( E \circ \text{hom}_\Sigma \). Thus, for all objects \( S \)
and $S'$ of $\Sigma$, $\text{hom}^\Sigma_{S} S S'$ is the discretely ordered predomain whose underlying set is $S \xrightarrow{\Sigma} S'$.

By taking the product of $\text{hom}^\Sigma_{S}$ and the identity functor on $K$ we obtain

$$\text{hom}^\Sigma_{S} \times 1_{K} \in \Sigma^{\text{op}} \times K \longrightarrow K \times K.$$  

We may compose this with the distinguished binary product functor for $K$, denoted

$$\Pi^{K}_{2} \in K \times K \longrightarrow K,$$

getting

$$\Pi^{K}_{2} \circ (\text{hom}^\Sigma_{S} \times 1_{K}) \in \Sigma^{\text{op}} \times K \longrightarrow K.$$  

The curried version of this last functor is denoted

$$H \in \Sigma^{\text{op}} \longrightarrow (K \Rightarrow K).$$

Hence if $S \in \text{Ob} \Sigma$ and $F \in \text{Ob} K$, then

$$H S F = (\text{hom}^\Sigma_{S}) \times_{K} F;$$

furthermore, if $T \in \text{Ob} \Sigma$, then

$$H S F T = (\text{hom}^\Sigma_{S T}) \times_{\text{Pdom}} (F T).$$

For each pair $F, F'$ of functors in $\Sigma \longrightarrow \text{Pdom}$ we seek to define

$$F \xrightarrow{K} F' = F \Rightarrow F' \in \Sigma \longrightarrow \text{Pdom}.$$  

Thus, for each object $S$ of $\Sigma$ we must describe a predomain $(F \Rightarrow F') S$. The underlying set of $(F \Rightarrow F') S$ is

$$\text{Ob} ((F \Rightarrow F') S) = H S F \xrightarrow{K} F' = ((\text{hom}^\Sigma_{S}) \times F) \xrightarrow{K} F',$$

which is a set of natural transformations. The partial order on $(F \Rightarrow F') S$ is given by

$$\eta_{S} \leq (F \Rightarrow F') S \eta' \text{ iff } \eta_{T} \leq _{H S F T \xrightarrow{\text{Pdom}}} F' \eta'_{T} \text{ for all } T \in \text{Ob} \Sigma.$$  

(Note that the underlying set of the predomain $H S F T \xrightarrow{\text{Pdom}} F' T$ is the set $H S F T \xrightarrow{\text{Pdom}} F' T$, of which both $\eta_{T}$ and $\eta'_{T}$ are members.) We must check that this partial order is directed-complete. Let $D$ be a directed subset of $(F \Rightarrow F') S$. The
definition of \( \leq \) \((F \Rightarrow F') S\) implies that, for each \( T \in \text{Ob } \Sigma\), \( DT\) is a directed subset of \( H S F T = F' T\). Hence, we can define an \((\text{Ob } \Sigma)\)-indexed collection \( \nu D \) of continuous functions by
\[
\nu D T = \bigcup_{HSFT = F'T} (DT).
\]
Of course, we aim to show that \( \nu D = \bigcup_{(F \Rightarrow F')} SD\). First we must verify that \( \nu D \in \text{Ob } ((F \Rightarrow F') S)\), i.e. \( \nu D \) is a natural transformation. Let \( \tau \in T \rightarrow T'\). We must show that
\[
(\text{hom}\Sigma ST \times (FT)) \xrightarrow{\nu D T} F'T
\]
commutes in \( P\text{dom} \). For each \( \eta \in D\), the commutativity of
\[
(\text{hom}\Sigma ST \times (FT)) \xrightarrow{\eta T} F'T
\]
implies that, whenever \( \sigma \in S \rightarrow T \) and \( t \in \text{Ob } (FT)\), we have
\[
\eta T' (\tau \circ \sigma, F't) = ((\eta T') \circ ((\text{hom}\Sigma ST \times (FT))) (\sigma, t)
\]
\[
= F'T (\eta T (\sigma, t)).
\]
Thus, we may write
\[
D T' (\tau \circ \sigma, F't) = F'T (\nu D T (\sigma, t)).
\]
Now take any \( \sigma \in S \rightarrow T \) and \( t \in \text{Ob } (FT)\), and compute:
\[
((\nu D T') \circ ((\text{hom}\Sigma ST \times (FT))) (\sigma, t)
\]
\[
= \nu D T' (\sigma \circ \tau, F't)
\]
\[
= \bigcup_{HSFT = F'T} (D T') (\sigma \circ \tau, F't)
\]
\[
= \bigcup_{F'T} (F'T (\nu D T (\sigma, t)))
\]
\[
= F'T (\bigcup_{F'T} (DT(\sigma, t)) ), \text{ by the continuity of } F'T,
\]
\[ F' \vdash (\mathcal{H}_{S^T \Rightarrow F'} T) (D T) \langle \sigma, t \rangle \]
\[ = F' T \langle \mathcal{V}_D T(\sigma, t) \rangle \]
\[ = ((F' T) \circ (\mathcal{V}_D T)) \langle \sigma, t \rangle, \]

proving \( \mathcal{V}_D \) is indeed a natural transformation. Next we will demonstrate that \( \mathcal{V}_D \) is actually the least upper bound of \( D \). Let \( \eta \in D \). For all \( T \in \text{Ob} \Sigma \), \( \eta_T \in D T \); consequently
\[ \eta_T \leq (\mathcal{H}_{S^T \Rightarrow F'} T) (D T) = \mathcal{V}_D T. \]

Thus, \( \eta \leq (F \Rightarrow F') S \mathcal{V}_D \), which shows \( \mathcal{V}_D \) is an upper bound of \( D \).

Suppose \( \mu \) is an upper bound of \( D \), so that for all \( \eta \in D \) and all \( T \in \text{Ob} \Sigma \), \( \eta_T \leq (\mathcal{H}_{S^T \Rightarrow F'} T) \mu T \). Then, for all \( T \in \text{Ob} \Sigma \),
\[ \mathcal{V}_D T = (\mathcal{H}_{S^T \Rightarrow F'} T) (DT) \leq (\mathcal{H}_{S^T \Rightarrow F'} T) \mu T \]

Thus \( \mathcal{V}_D \leq (F \Rightarrow F') S \mu \), which demonstrates that \( \mathcal{V}_D \) is the least upper bound of \( D \). This completes the proof that \( (F \Rightarrow F') S \) is a predomain.

Now we turn to the task of defining \( F \Rightarrow F' \) on morphisms of \( \Sigma \). Let \( \sigma \in S \rightarrow S' \). We must give a continuous function.
\[ (F \Rightarrow F') \sigma \in (F \Rightarrow F') S \rightarrow (F \Rightarrow F') S'. \]
So take \( \eta \in ((\text{hom}_\Sigma S) \times F) \rightarrow F' \); we must define
\[ K (F \Rightarrow F') \sigma \eta \in ((\text{hom}_\Sigma S') \times F) \rightarrow F'. \]

Let \( T \in \text{Ob} \Sigma \). The function
\[ (F \Rightarrow F') \sigma \eta \in ((\text{hom}_\Sigma S' T) \times (F T)) \rightarrow (F' T) \]

is given by
\[ (F \Rightarrow F') \sigma \eta \in \tau \rightarrow \eta \langle \tau \circ \sigma, t \rangle \]
where \( \tau \in S' \rightarrow T \) and \( t \in \text{Ob} (F T) \). There are a number of points that need verification. First, we will show that the function \( (F \Rightarrow F') \sigma \eta T \) is continuous. Because the partial order on the first component is discrete, each directed subset of \( (\text{hom}_\Sigma S' T) \times (F T) \) has the form \( \{ (\tau, t) \mid t \in D \} \) where
\( \tau \in S' \rightarrow T \) and \( D \) is a directed subset of \( F \cdot T \), and for each such pair \( \tau, D \) we have the following computation:

\[
(F \Rightarrow F') \circ \eta_T \left( \bigcup_{\text{HS}_{FT}} \{ \langle \tau, t \rangle \mid t \in D \} \right) = (F \Rightarrow F') \circ \eta_T (\tau, \bigcup_{\text{PT}_D} D)
\]

\[
= \eta_T (\tau \circ \sigma, \bigcup_{\text{PT}_D} D)
\]

\[
= \eta_T \left( \bigcup_{\text{HS}_{FT}} \{ \langle \tau \circ \sigma, t \rangle \mid t \in D \} \right)
\]

\[
= \bigcup_{F \cdot T} (\eta_T (\tau \circ \sigma, t) \mid t \in D), \text{ since } \eta_T \text{ is continuous},
\]

\[
= \bigcup_{F \cdot T} \{ (F \Rightarrow F') \circ \eta_T (\tau, t) \mid t \in D \}.
\]

Hence, \( (F \Rightarrow F') \circ \eta_T \) is indeed continuous. Second, we must show that \( (F \Rightarrow F') \circ \eta_T \) is a natural transformation. Let \( \tau \in T \rightarrow T' \).

We must demonstrate that

\[
\begin{array}{c}
\text{commutes in } \text{Pd} \text{om}. \text{ We will use the commutativity of}
\end{array}
\]

\[
\begin{array}{ccc}
\text{hom}_{\Sigma}^{S'} T \times (F T) & \overset{(F \Rightarrow F') \circ \eta_T}{\longrightarrow} & F' T
\\
\text{hom}_{\Sigma}^{S'} T \times (F T) & \overset{\eta_T}{\longrightarrow} & F' T
\\
\text{hom}_{\Sigma}^{T' T} \times (F' T) \end{array}
\]

which is a consequence of the naturality of \( \eta \). Let \( \rho \in S' \rightarrow T \) and \( t \in \text{Ob} (F T) \). Then

\[
(F \Rightarrow F') \circ \eta_T' \left( \left( \text{hom}_{\Sigma}^{S' T} \times (F T) \right) (\rho, t) \right) = (F \Rightarrow F') \circ \eta_T' (\tau \circ \rho, F t)
\]

\[
= \eta_T' (\tau \circ \rho \circ \sigma, F t)
\]

\[
= \eta_T' \left( \left( \text{hom}_{\Sigma}^{S T} \times (F T) \right) (\rho \circ \sigma, t) \right)
\]

\[
= F' T (\eta_T (\rho \circ \sigma, t))
\]

\[
= F' T ((F \Rightarrow F') \circ \eta_T (\rho, t)),
\]

as desired. Finally we must check that \( (F \Rightarrow F') \circ \) is continuous.
Let $D$ be a directed subset of $(F \to F') S$. We will show that

$$(F \to F') \circ (\bigsqcup_{(F \to F') S} D) = \bigsqcup_{(F \to F') S'} ((F \to F') \circ D).$$

Let $T \in \text{Ob } S', \tau \in S' \longrightarrow T, t \in \text{Ob } (FT)$. Then

$$(F \to F') \circ (\bigsqcup_{(F \to F') S} D) T (\tau, t) = (\bigsqcup_{(F \to F') S} D) T (\tau \circ \sigma, t) = (\bigsqcup_{HST = F' T} (DT)) (\tau \circ \sigma, t) = \bigsqcup_{F' T} (DT (\tau \circ \sigma, t)) = \bigsqcup_{F' T} ((F \to F') \circ DT (\tau, t)) = \bigsqcup_{HST = F' T} ((F \to F') \circ DT) (\tau, t) = \bigsqcup_{(F \to F') S'} ((F \to F') \circ D) T (\tau, t)$$

from which the desired result follows immediately.

Now that we have defined $F \to F'$ on both objects and arrows of $\Sigma$, we must verify that it is a functor by showing that $F \to F'$ preserves composition and identity arrows. Let $\sigma \in S \longrightarrow S', \sigma' \in S' \longrightarrow S''$. The equalities

$$(F \to F') \circ (\sigma' \circ \sigma) = ((F \to F') \circ \sigma') \circ ((F \to F') \circ \sigma)$$

and

$$(F \to F') L^\Sigma_\sigma = L^P^\text{dom}_{(F \to F') S}$$

follow from the computations

$$(F \to F') (\sigma' \circ \sigma) \eta T (\tau, t) = \eta T (\tau \circ \sigma' \circ \sigma, t) = (F \to F') \circ \eta T (\tau \circ \sigma', t) = (F \to F') \circ \sigma' ((F \to F') \circ \eta T (\tau, t)) = (((F \to F') \circ \sigma') \circ ((F \to F') \circ \sigma)) \eta T (\tau, t)$$

and

$$(F \to F') L^\Sigma_\sigma \eta T (\rho, t) = \eta T (\rho \cdot L^\Sigma_\sigma, t) = \eta T (\rho, t) = L^P^\text{dom}_{(F \to F') S} \eta T (\rho, t),$$
where \( \eta \in (\text{hom}^E S) \times F \xrightarrow{\kappa} F', T \in \text{Ob } \Sigma, \)

\( T \in S' \xrightarrow{\Sigma} T, t \in \text{Ob } (FT), \) and \( \rho \in S \xrightarrow{\Sigma} T. \) This completes
the definition of \( F \Rightarrow F' \) and the proof that it is a functor.

Given functors \( F, F' \in \Sigma \xrightarrow{\Sigma} \text{Pdom}, \) we now turn to the
definition of the natural transformation

\( Ap = Ap (F, F') \in (\text{hom}^E \Sigma(S, T)) \times \text{Pdom}(F, S) \xrightarrow{\text{Pdom}} F'S \)

Let \( S \in \text{Ob } \Sigma. \) Then

\( Ap S \in (\text{hom}^E \Sigma(S, T)) \times \text{Pdom}(F, S) \xrightarrow{\text{Pdom}} F'S \)
is given by

\( Ap S \langle \eta, s \rangle = \eta S \langle l^E_S, s \rangle \)

where \( \eta \in (\text{hom}^E \Sigma(S, T)) \times \text{Pdom}(F, S) \xrightarrow{\text{Pdom}} F' \) and \( s \in \text{Ob } (FS). \)

Some verifications are in order. First, we must check that
Ap S is continuous, i.e. if \( D \) denotes a directed subset of
\( (\text{hom}^E \Sigma(S, T)) \times \text{Pdom}(F, S), \) then

\( Ap S \cup ((F \Rightarrow F') S) \times (FS) = \bigcup F'S \langle Ap S D \rangle. \)

A consequence of the continuity of the projection mappings from
a product onto its factors is that the directed subset \( D \) gives
rise to directed subsets

\( D_1 = \{ \eta \mid \langle \eta, s \rangle \in D \}, \quad D_2 = \{ s \mid \langle \eta, s \rangle \in D \} \) and \( D' = D_1 \times D_2 \)

Also,

\( \bigcup ((F \Rightarrow F') S) \times (FS) = \bigcup (F \Rightarrow F') S D_1 \cup FS D_2 = \bigcup ((F \Rightarrow F') S) \times (FS) D'. \)

A routine argument establishes that the least upper bounds of
the two sets

\( Ap S D = \{ \eta S \langle l^E_S, s \rangle \mid \langle \eta, s \rangle \in D \} \)

\( Ap S D' = \{ \eta S \langle l^E_S, s \rangle \mid \langle \eta, s \rangle \in D' \} \)
are the same. Now compute:

\[
\begin{align*}
\text{Ap } S \ (\text{L}_1 \ ((F \Rightarrow F') S) \times (FS) D) \\
= \text{Ap } S \ (\text{L}_1 \ (F \Rightarrow F') S D_1, \ U_{FS} D_2) \\
= (\text{L}_1 (F \Rightarrow F') S D_1) \ (l^\Sigma, \ U_{FS} D_2) \\
= (\text{L}_{HSFS \Rightarrow F' S} (D_1 S)) \ (l^\Sigma, \ U_{FS} D_2) \\
= \text{L}_{F' S} \ {\eta S} \ (l^\Sigma, \ U_{FS} \ {s \mid s \in D_2}) \ | \ {\eta \in D_1} \\
= \text{L}_{F' S} \ {\eta S} \ (l^\Sigma, s) \ | \ (\eta, s) \in D' \\
= \text{L}_{F' S} \ (\text{Ap } S \ D') \\
= \text{L}_{F' S} \ (\text{Ap } S \ D),
\end{align*}
\]

as desired. Second, we must verify that Ap is a natural transformation. Suppose \( \sigma \in S \xrightarrow{\Sigma} S' \). We must establish the commutativity in \( P\text{dom} \) of

\[
\begin{array}{ccc}
((F \Rightarrow F') S) \times (FS) & \xrightarrow{\text{Ap } S} & F'S \\
((F \Rightarrow F') S) \times (FS) \xrightarrow{\Sigma} (F \sigma) & \xrightarrow{\text{Ap } S'} & F'S' \\
((F \Rightarrow F') S) \times (FS') & \xrightarrow{\Sigma} (F \sigma) & F'S'
\end{array}
\]

Let \( \eta \in \left( \text{hom}_S^\Sigma \ (S) \times F \right) \xrightarrow{\Sigma} F' \) and \( s \in \text{Ob} (FS) \).

We know that

\[
\begin{align*}
(\text{hom}_S^\Sigma \ (S) \times (FS)) \xrightarrow{\eta S} F'S \\
(\text{hom}_S^\Sigma \ (S) \sigma) \xrightarrow{\Sigma} (F \sigma) & \xrightarrow{\eta S'} F'S'
\end{align*}
\]

commutes in \( P\text{dom} \). Compute:

\[
\begin{align*}
((\text{Ap } S')) \circ (((F \Rightarrow F') \sigma) \times (F \sigma)) \ (\eta, s) \\
= \text{Ap } S' \ (((F \Rightarrow F') \sigma) \times (F \sigma)) \ (\eta, s) \\
= \text{Ap } S' \ ((F \Rightarrow F') \sigma \eta, F \sigma s)
\end{align*}
\]
= (F \Rightarrow F') \circ \eta \ S' (l^\Sigma_S, F \circ s)
= \eta S' (l^\Sigma_S, \circ \sigma, F \circ s)
= \eta S' (\sigma \circ l^\Sigma_S, F \circ s)
= \eta S' ((\text{hom}^\Sigma_S \circ (F \circ \sigma)) \times (l^\Sigma_S, s))
= F' \circ (\eta S (l^\Sigma_S, s))
= F' \circ (\text{Ap} \ S (\eta, s))
= ((F' \circ \sigma) \circ (\text{Ap} \ S)) (\eta, s).

Therefore, \text{Ap} = \text{Ap} (F, F') is indeed a natural transformation.

Given functors F, F', F'' in \xymatrix{\Sigma \ar[r] & \text{Pdom}} our task is to define

\text{Ab} = \text{Ab} (F, F', F'') \in ((F \times F') \ar[r] & F'') \ar[r] & (F \ar[r] & (F' \Rightarrow F'')) \ar[r] & \text{Set} \ar[r] & K

Take a natural transformation \eta \in (F \times F') \ar[r] & F''; we want a natural transformation

\text{Ab} \ \eta \in F \ar[r] & (F' \Rightarrow F'').

So let S \in \text{Ob} \Sigma. We must define

\text{Ab} \ \eta \ S \in P \ar[r] & (F' \Rightarrow F'') \ S.

Let s \in \text{Ob} \ (FS); we need

\text{Ab} \ \eta \ S \ s \in (\text{hom}^\Sigma_S) \ x \ F' \ar[r] & F''.

Let T \in \text{Ob} \Sigma. Define

\text{Ab} \ \eta \ S \ s \ T \in (\text{hom}^\Sigma_S \ T) \ x (F' \ T) \ar[r] & F'' \ T

by

\text{Ab} \ \eta \ S \ s \ T (\sigma, t) = \eta T (F s \circ t)

where \sigma \in S \ar[r] & T and t \in \text{Ob} (F' \ T). Of course, each layer of this definition requires some sort of verification. First, we must check that \text{Ab} \ \eta \ S \ s \ T is continuous. Since \text{hom}^\Sigma_S \ T is discretely ordered, an arbitrary directed subset of (\text{hom}^\Sigma_S \ T)
\( \times (F'T) \) may be described as \( \{ (\sigma, t) \mid t \in D \} \) where \( \sigma \in S \rightarrow T \) and \( D \) is a directed subset of \( F'T \). The computation

\[
\begin{align*}
\text{Ab } \eta \ s \ s \ T \ \cup \ \text{HSP}_T \{ (\sigma, t) \mid t \in D \} &= \text{Ab } \eta \ s \ s \ T \ (\sigma, \cup \ FP'TD) \\
&= \eta_T (F \sigma s, \cup FP'TD) \\
&= \eta_T (\cup (FT) \times (F'T) \{ (F \sigma s, t) \mid t \in D \}) \\
&= \cup F'T'' \{ \eta_T (Fos, t) \mid t \in D \}, \text{ by continuity of } \eta_T, \\
&= \cup F'T'' \{ \text{Ab } \eta \ s \ s \ T (\sigma, t) \mid t \in D \}
\end{align*}
\]

demonstrates that \( \text{Ab } \eta \ s \ s \ T \) is continuous. Second, we must check that \( \text{Ab } \eta \ s \ s \ s \) is a natural transformation. Let

\( \tau \in T \rightarrow T' \). We need to demonstrate the commutativity of

\[
\begin{array}{ccc}
\text{Ab } \eta \ s \ s \ T & \xrightarrow{(\text{hom}_T S T) \times (F'T)} & F'T' \\
\downarrow & & \downarrow \text{F'T'} \\
\text{Ab } \eta \ s \ s \ T' & \xrightarrow{(\text{hom}_T S T') \times (F'T')} & \text{F'T'}
\end{array}
\]

Let \( \sigma \in S \rightarrow T \) and \( t \in \text{Ob } (F'T) \), compute:

\[
\begin{align*}
((\text{Ab } \eta \ s \ s \ T')) \circ ((\text{hom}_T S T) \times (F'T)) (\sigma, t) &= \text{Ab } \eta \ s \ s \ T' ((\text{hom}_T S T) \times (F'T)) (\sigma, t) \\
&= \text{Ab } \eta \ s \ s \ T' (\tau \sigma, F'Tt) \\
&= \eta_T' (F (\tau \sigma) s, F'Tt) \\
&= \eta_T' (FT (Fos), F'Tt), \text{ because } F \text{ is a functor,} \\
&= \eta_T' ((FT) \times (F'T)) (Fos, t) \\
&= F'T (\eta_T (Fos, t)), \text{ by the naturality of } \eta, \\
&= F'T (\text{Ab } \eta \ s \ s \ T (\sigma, t)) \\
&= ((\text{F'T}) \circ (\text{Ab } \eta \ s \ s \ T)) (\sigma, t).
\end{align*}
\]

Thus \( \text{Ab } \eta \ s \ s \ ) is natural. Third, we will check that \( \text{Ab } \eta \ s \)
is continuous. Let $D$ be a directed subset of $FS$. If 
$T \in \text{Ob } \Sigma$, $\sigma \in S \longrightarrow T$, and $t \in \text{Ob } (F'T)$, then 

$$\text{Ab } \eta \; S \; \left( \bigcup_{FS} D \right) \; T \; (\sigma, t)$$

$$= \eta T \left( F \; \sigma \; \left( \bigcup_{FS} D \right), \; t \right)$$

$$= \eta T \left( \bigcup_{F' T} (F \sigma D), \; t \right), \text{ because } F \sigma \text{ is continuous,}$$

$$= \eta T \left( \bigcup_{(FT) \times (F'T)} \{(F \sigma s, \; t) \mid s \in D \} \right)$$

$$= \bigcup_{F' T} \eta \left( F \sigma s, \; t \right), \text{ because } \eta T \text{ is continuous,}$$

$$= \bigcup_{(F' \Rightarrow F'')} (\text{Ab } \eta \; S \; D \; T \; (\sigma, \; t))$$

$$= \left( \bigcup_{(FS') \Rightarrow (F' \Rightarrow F'')} \text{Ab } \eta \; S \; D \; (\sigma, \; t) \right) \text{ (Ab } \eta \; S \; D \; T \; (\sigma, \; t))$$

Therefore,

$$\text{Ab } \eta \; S \; \left( \bigcup_{FS} D \right) = \bigcup_{(F' \Rightarrow F'')} (\text{Ab } \eta \; S \; D),$$

which shows $\text{Ab } \eta \; S$ is continuous. Finally we must prove that $\text{Ab } \eta$ is a natural transformation. Suppose $\sigma \in \Sigma \longrightarrow S'$. 

We are interested in establishing the commutativity in $\text{Pdom}$ of

$$\begin{array}{ccc}
FS & \xrightarrow{\text{Ab } \eta \; S} & (F' \Rightarrow F'') \; S \\
\downarrow \text{F} \sigma & & \downarrow (F' \Rightarrow F'') \sigma \\
FS' & \xrightarrow{\text{Ab } \eta \; S'} & (F' \Rightarrow F'') \; S'
\end{array}$$

Let $s \in \text{Ob } (FS)$, $T \in \text{Ob } \Sigma$, $\tau \in S' \longrightarrow T$, and $t \in \text{Ob } (F'T)$. Then 

$$((\text{Ab } \eta \; S')) \; \circ \; (F \sigma) \; s \; T \; (\tau, \; t)$$

$$= \text{Ab } \eta \; S' \; (F \sigma s) \; T \; (\tau, \; t)$$

$$= \eta T \left( F \; \sigma \; (F \sigma s), \; t \right)$$

$$= \eta T \left( F \; (\tau \sigma) \; s, \; t \right), \text{ because } F \text{ is a functor,}$$

$$= \text{Ab } \eta \; S \; s \; T \; (\tau \sigma, \; t)$$
\[ = (F' \Rightarrow F") \circ (\text{Ab } \eta \ S \ s) \ T (\tau, t) \]
\[ = (((F' \Rightarrow F") \circ) \circ (\text{Ab } \eta \ S)) \ s \ T (\tau, t), \]

and from this computation commutativity of the diagram above is clear.

We now arrive at the heart of the proof that \( K \) is a Cartesian closed category. Given functors \( F, F', F'' \) in \( \Sigma \rightarrow \text{Pdom} \), we claim that for each natural transformation \( \eta \in F \times F' \xrightarrow{K} F'' \) the natural transformation \( \text{Ab } \eta \in F \xrightarrow{K} (F' \Rightarrow F'') \) is the unique arrow making the diagram

\[
\begin{array}{ccc}
F \times F' & \xrightarrow{(\text{Ab } \eta) \times 1_{F'}} & (F' \Rightarrow F'') \times F' \\
\downarrow \eta & & \downarrow \text{Ap} = \text{Ap}(F', F'') \\
F'' & &
\end{array}
\]

commute in \( K \). Let \( S \in \text{Ob } \Sigma, s \in \text{Ob } (FS), s' \in \text{Ob } (F'S) \).

Then
\[
(\text{Ap} \circ ((\text{Ab } \eta) \times 1_{F'})) \ S \ (s, s')
\]
\[ = ((\text{Ap } S) \circ ((\text{Ab } \eta \ S) \times (1_{F'}^{\text{Pdom}}))) \ (s, s') \]
\[ = \text{Ap } S (((\text{Ab } \eta \ S) \times 1_{F'}^{\text{Pdom}})(s, s')) \]
\[ = \text{Ap } S (\text{Ab } \eta \ S \ s, s') \]
\[ = \text{Ab } \eta \ S \ s \ s' \]
\[ = \eta S (F \eta \ S \ s, s') \]
\[ = \eta S (s, s'), \text{ because F is a functor.} \]

Thus, the diagram commutes. To deal with the uniqueness assertion, suppose \( \theta \in F \xrightarrow{K} (F \Rightarrow F'') \) and the diagram
commutes. Let \( S \in \text{Ob} \Sigma, s \in \text{Ob} (F S), T \in \text{Ob} \Sigma, \sigma \in S \xrightarrow{\Sigma} T, \) and \( t \in \text{Ob} (F' T). \) We compute as follows:

\[
\theta \ S \ s \ T \ < \sigma, \ t > \\
= (F' \Rightarrow F'') \sigma \ (\theta s) \ T \ < \xi^T, t > , \text{ by the definition of } F \Rightarrow F', \\
= (((F' \Rightarrow F'') \sigma) \circ (\theta s)) \ s \ T \ < \xi^T, t > \\
= ((\theta T) \circ (F s)) \ s \ T \ < \xi^T, t > , \text{ by the naturality of } \theta , \\
= \theta T (F s) \ T \ < \xi^T, t > \\
= \text{Ap} \ T \ < \theta T (F s), t > , \text{ by the definition of Ap}, \\
= \text{Ap} \ T \ < \theta T (F s), t > \\
= \text{Ap} \ T \ < (\theta T) \times (1_{F'T}) \ (F s, t) \\
= \theta T (F s, t), \text{ by the above diagram,} \\
= \text{Ab} \ \eta \ S \ s \ T \ < \sigma, t > , \text{ by the definition of } \text{Ab} \ \eta .
\]

Therefore, \( \theta = \text{Ab} \ \eta, \) and this proves the claim. \( \square \)

The category \( \Sigma \Rightarrow \text{Pdom} \) is a little bit too large for the applications to the semantics of programming languages that we have in mind; its use would involve many annoying verifications that certain predomains are indeed domains. Thus, we might try using \( \Sigma \Rightarrow \text{Dom} \) rather than \( \Sigma \Rightarrow \text{Pdom}. \) However, the obvious way of showing that \( \Sigma \Rightarrow \text{Dom} \) is a Cartesian closed category, i.e. by embedding it in \( \Sigma \Rightarrow \text{Pdom}, \) fails! We suspect
that, for some $\Sigma$, $\Sigma \Rightarrow \mathcal{D}om$ cannot be made into a Cartesian closed category, although we are unable at this time to give a proof. Another idea is to use $\Sigma \Rightarrow S\mathcal{D}om$. This, too, is doomed, as one can see by taking $\Sigma$ to be the category with one object and one arrow; then $\Sigma \Rightarrow S\mathcal{D}om$ is isomorphic to $S\mathcal{D}om$, which we have shown is not Cartesian closed. Fortunately, there is another alternative, which we now present.

Let $M$ be the full subcategory of $K = \Sigma \Rightarrow P\mathcal{D}om$ whose collection of objects consists of those functors which factor through the inclusion functor $J : S\mathcal{D}om \rightarrow P\mathcal{D}om$. Thus a functor $F : \Sigma \rightarrow P\mathcal{D}om$ is an object of $M$ if (1) $FS$ is a domain for every $S \in \text{Ob } \Sigma$, and (2) $F\sigma$ is a strict continuous function for every $\sigma \in \text{Ar } \Sigma$. The inclusion functor $J$ induces an embedding

$\Sigma \Rightarrow J \in \Sigma \Rightarrow S\mathcal{D}om \rightarrow \Sigma \Rightarrow P\mathcal{D}om$

given by

$(\Sigma \Rightarrow J) \ F = J \circ F$, for $F \in \text{Ob } (\Sigma \Rightarrow S\mathcal{D}om),$

and

$(\Sigma \Rightarrow J) \ \eta = J \star \eta$, for $\eta \in \text{Ar } (\Sigma \Rightarrow S\mathcal{D}om).$

At first glance, it might appear that the image of $\Sigma \Rightarrow J$ is $M$. However, this is false, because the assertion $\eta \in F \Rightarrow G$ requires that $\eta S$ be a continuous function for each $S \in \text{Ob } \Sigma$,

whereas $\eta \in F \Rightarrow G$ corres-

pondingly requires that $\eta S$ be a strict, continuous function for each $S \in \text{Ob } \Sigma$. 
It is easy to see that the distinguished product functors for $K$ give objects of $M$ when applied to objects of $M$, and that the terminal object of $K$ is an object of $M$. These observations are the groundwork for the following theorem.

**Theorem 5.5:** The category $M$ is Cartesian closed, with $\Rightarrow$, $\text{Ap}$, and $\text{Ab}$ defined by restricting to $M$ the corresponding entities for $K$.

**Proof:** Let the functors $F$ and $G$ be objects of $M$. It suffices to show that $F \xrightarrow{K} G$ is an object of $M$.

Suppose $S \in \text{Ob } \Sigma$. We wish to show that the predomain $(F \xrightarrow{K} G)S$ has a minimal element. Recall that the underlying set of this predomain is the set of natural transformations

$$(\text{hom}_\Sigma S) \times F \xrightarrow{K} G.$$

We attempt to define a natural transformation $\mu$ in this set by

$$\mu_T(\sigma, t) = \lambda_{GT}$$

where $T \in \text{Ob } \Sigma$, $\sigma \in S \xrightarrow{E} T$, $t \in \text{Ob } (FT)$. For each $T \in \text{Ob } \Sigma$,

$$\mu_T \in (\text{hom}_\Sigma S T) \times (F T) \xrightarrow{P\text{dom}} GT,$$

because $\mu_T$ is a constant function and, hence, a continuous function. Naturality of $\mu$ follows from consideration of the diagram.
(hom\(_\Sigma\) S T) × (FT) \xrightarrow{\mu_T} GT
(hom\(_\Sigma\) S T) × (FT) \xrightarrow{\mu_{T'}} GT'

\(\sum_{T \in T} S \rightarrow T'\); the diagram commutes because \(G_T\) is
strict. (This is why the definition of \(M\) involves \(S_{dom}\)
rather than \(D_{om}\).) Thus
\[\mu \in \text{hom}\(\Sigma\) S × F \xrightarrow{K} G.\]

The fact that \(\mu\) is minimal in \((F \Rightarrow G) S\) follows from
the minimality of \(\mu_T\) in \(H F S T \xrightarrow{\text{Pdom}} GT\). Therefore we
may legitimately write
\[\mu = \bot_{(F \Rightarrow G) S}^K.\]

Suppose \(\sigma \in S \xrightarrow{\Sigma} S'\). We aim to show that
\((F \Rightarrow G) \sigma\) is strict. (We know it is continuous.)

Let \(\tau \in S' \xrightarrow{\Sigma} T, t \in \text{Ob} (F T)\). Then
\(\sigma \in \text{hom}\(\Sigma\) S \xrightarrow{\sum} T (\tau, t)\)
\[\Rightarrow \bot_{(F \Rightarrow G) S}^K T (\tau \circ \sigma, t), \text{ by definition of } (F \Rightarrow G) \sigma,\]
\[\Rightarrow \bot_{GT} T (\tau, t),\]
\[\Rightarrow \bot_{(F \Rightarrow G) S'} T (\tau, t).\]

Hence
\[\sigma \in \text{hom}\(\Sigma\) (F \Rightarrow G) S \Rightarrow \bot_{(F \Rightarrow G) S} = \bot_{(F \Rightarrow G) S'}.\]

We conclude \((F \Rightarrow G) K\) is an object of \(M\), as desired. \(\square\)
CHAPTER VI
A CATEGORY OF STORE SHAPES

So far we have been vague about the structure of $\Xi$, the category of store shapes. Obviously, the semantics of a language can be precisely described only if $\Xi$ is completely specified. There are actually a variety of choices for $\Xi$. As one approach, we may posit the existence of a set of "locations" along with operations for creating and manipulating "stacks of locations," which are the objects of $\Xi$. This approach has a very operational flavor. It has two drawbacks. First, the details turn out to be very complicated, and, second, we are uncomfortable with insisting at the outset that stores are "stacks of locations." Another candidate for $\Xi$ is the category of store shapes and expansions described in [6]. It avoids the drawbacks of the operational approach, but it does not seem possible to give continuation semantics by using that category for $\Xi$.

The point of this section is to describe a category $\Xi$ that we feel is at the proper level of abstraction. There is behind it an accessible intuition. At the same time we can avoid unnecessary complexities that may almost be said to be implementation-dependent (i.e., what kind of data type values can be placed in what locations, and what do we do with values that need several locations.)
The basic idea is to let any set \( X \) be an object of \( \Sigma \) (i.e. a store shape); then the set of stores corresponding to the store shape \( X \) is just \( X \) itself. An expansion in \( X \xrightarrow{\epsilon} Y \) is supposed to go from a "little" store shape \( X \) to a "big" store shape \( Y \). Part of an expansion is a "forgetting" function \( \phi \in Y \times X \), which allows a more complicated store to serve as a less complicated one. Another part of an expansion is a "replacement" function \( \rho \in X \times (Y \times Y) \); it is used when one wishes to overwrite the part of a more complicated store that corresponds to a less complicated store. Thus an arrow of \( \Sigma \) is an ordered pair

\[
\langle \phi, \rho \rangle \in X \xrightarrow{\epsilon} Y,
\]

where

\[
\phi \in Y \times X \quad \text{and} \quad \rho \in X \times (Y \times Y).
\]

Furthermore, we require that an arrow \( \langle \phi, \rho \rangle \in X \xrightarrow{\epsilon} Y \) satisfy the following three conditions:

\[
\begin{align*}
(\Sigma 1) \quad & \phi \ (\rho \times y) = x, \\
(\Sigma 2) \quad & \rho \ (\phi \ y) \ y = y, \ \text{and} \\
(\Sigma 3) \quad & \rho \times (\rho \times y) = \rho \times y,
\end{align*}
\]

where \( x, x' \in X \) and \( y \in Y \). We may read \((\Sigma 1)\) loosely as replacing with \( x \) in \( y \) and then forgetting gives back \( x \). Condition \((\Sigma 2)\) says replacing in \( y \) with part of itself gives back \( y \). Finally, \((\Sigma 3)\) says only the last replacement counts. We could obtain a larger and simpler category by
not imposing these three conditions on morphisms, and all of the results of this chapter would hold, with the exception of the theorem on the structure of morphisms. However, the demonstrations of naturality in Chapters VII and VIII seem to require (Σ1), (Σ2), and (Σ3) or some similar conditions.

In order to define composition elegantly in Σ we need some notation. Suppose α ∈ X → Y and β ∈ W → Z. Then

\[ \alpha \circ \beta \in (Y \circ W) \circ (X \circ Z) \]

is given by

\[ (\alpha \circ \beta) \circ f = \beta \circ f \circ \alpha, \]

where \( f \in Y \circ W \). (So \( \circ \) is just the ordinary hom functor for Set.) Also, for each set \( X \), define the diagonalization \( D_X \) for \( X \) to be

\[ D_X \in (X \circ (X \circ X)) \circ (X \circ X), \]

given by

\[ D_X \circ f \circ x = f \circ x \]

where \( f \in X \circ (X \circ X) \) and \( x \in X \).

Suppose

\[ (\phi, \rho) \in X \xrightarrow{\Sigma} Y \text{ and } (\phi', \rho') \in Y \xrightarrow{\Sigma} Z. \]

Define

\[ (\phi', \rho') \circ (\phi, \rho) \in X \xrightarrow{\Sigma} Z \]
by

\[ (\phi', \rho') \circ (\phi, \rho) = (\phi \circ \phi', D_Z \circ (\phi' \rightarrow \rho') \circ \rho) \]

Obtaining a new forgetting function by composing given forgetting functions is not surprising. The second component on the right-hand side is the functional composition

\[ X \xrightarrow{\rho} (Y \rightarrow Y) \xrightarrow{\phi' \rightarrow \rho'} (Z \rightarrow (Z \rightarrow Z)) \xrightarrow{D_Z} (Z \rightarrow Z). \]

For \( x \in X, z \in Z \), we have

\[
(D_Z \circ (\phi' \rightarrow \rho') \circ \rho) x z = D_Z ((\phi' \rightarrow \rho') (\rho x)) z
= D_Z (\rho' \circ (\rho x) \circ \phi') z
= (\rho' \circ (\rho x) \circ \phi') z z
= \rho' (\rho x (\phi' z)) z.
\]

Thus, we have an intuitively satisfying replacement function derived from two expansions. Given that \((\phi, \rho)\) and \((\phi', \rho')\) satisfy the three conditions, it is necessary to check that their composite also satisfies them. Let \( x, x' \in X \) and \( z \in Z \). For \((\Sigma 1)\), we have

\[
(\phi \circ \phi') ((D_Z \circ (\phi' \rightarrow \rho') \circ \rho) x z )
= (\phi \circ \phi') (\rho' (\rho x (\phi' z)) z)
= \phi (\phi' (\rho x (\phi' z)) z)
= \phi (\rho x (\phi' z)), \text{ by } (\Sigma 1) \text{ for } (\phi', \rho').
= x, \text{ by } (\Sigma 1) \text{ for } (\phi, \rho).
\]
For $(\Sigma 2)$, we compute

\[
(D' \circ (\phi' + \rho') \circ \rho) ((\phi \circ \phi') \circ \rho') \circ \rho = (D' \circ (\phi' + \rho') \circ \rho) ((\phi \circ \phi') \circ \rho') \circ \rho
\]

\[
= \rho' ((\phi \circ \phi') \circ \rho') \circ \rho \circ (\phi' \circ \rho' \circ \rho') \circ \rho
\]

\[
= \rho' ((\phi' \circ \rho') \circ \rho \circ (\phi' \circ \rho') \circ \rho) \circ \rho
\]

\[
= \rho' ((\phi' \circ \rho') \circ \rho \circ (\phi' \circ \rho') \circ \rho) \circ \rho, \quad \text{by (\Sigma 2) for } \langle \phi', \rho' \rangle
\]

Finally, for $(\Sigma 3)$, we have

\[
(D' \circ (\phi' + \rho') \circ \rho) \times (D' \circ (\phi' + \rho') \circ \rho) \times (\phi' \circ \rho') \circ \rho
\]

\[
= (D' \circ (\phi' + \rho') \circ \rho) \times (\rho' \times (\phi' \circ \rho') \circ \rho) \times (\phi' \circ \rho') \circ \rho
\]

\[
= \rho' ((\phi' \circ \rho') \circ \rho \times (\phi' \circ \rho') \circ \rho) \times (\phi' \circ \rho') \circ \rho
\]

\[
= \rho' ((\phi' \circ \rho') \circ \rho \times (\phi' \circ \rho') \circ \rho) \times (\phi' \circ \rho') \circ \rho, \quad \text{by (\Sigma 1) for } \langle \phi', \rho' \rangle
\]

\[
= \rho' \times (\phi' \circ \rho') \times (\phi' \circ \rho') \times (\phi' \circ \rho') \circ \rho
\]

\[
= \rho' \times (\phi' \circ \rho') \circ \rho, \quad \text{by (\Sigma 3) for } \langle \phi', \rho' \rangle
\]

\[
= \rho' \times (\phi' \circ \rho') \times \rho, \quad \text{by (\Sigma 3) for } \langle \phi', \rho' \rangle
\]

\[
= (D' \circ (\phi' + \rho') \circ \rho) \times \rho
\]

Note the curious fact that both $(\Sigma 1)$ and $(\Sigma 3)$ play roles in the last computation.

We must check that composition in $\Sigma$ is associative.

Suppose

\[
\langle \phi_1, \rho_1 \rangle \in X \longrightarrow Y, \langle \phi_2, \rho_2 \rangle \in Y \longrightarrow Z, \langle \phi_3, \rho_3 \rangle \in Z \longrightarrow W.
\]
First we need to verify that

\[ D_W \circ (\phi_3 \circ \rho_3) \circ D_Z = D_W \circ (\phi_3 \circ (D_W \circ (\phi_3 \circ \rho_3))) \]

both sides are in \((Z \times (Z \times Z)) \times (W \times W)\). Let \(h \in Z \times (Z \times Z), w, w' \in W\). Then

\[
((\phi_3 \circ \rho_3) \circ D_Z) \ h \ w \ w'
\]

\[
= (\phi_3 \circ \rho_3) \ (D_Z \ h) \ w \ w'
\]

\[
= (\rho_3 \circ (D_Z \ h) \circ \phi_3) \ w \ w'
\]

\[
= \rho_3 \ (D_Z \ h \ (\phi_3 \ w)) \ w'
\]

\[
= \rho_3 \ (h \ (\phi_3 w) \ (\phi_3 w)) \ w'
\]

and

\[
(\phi_3 \circ (D_W \circ (\phi_3 \circ \rho_3))) \ h \ w \ w'
\]

\[
= (D_W \circ (\phi_3 \circ \rho_3) \circ h \circ \phi_3) \ w \ w'
\]

\[
= D_W \ ((\phi_3 \circ \rho_3) (h \ (\phi_3 w))) \ w'
\]

\[
= D_W \ (\rho_3 \circ (h \ (\phi_3 w)) \circ \phi_3) \ w'
\]

\[
= (\rho_3 \circ (h \ (\phi_3 w)) \circ \phi_3) \ w' \ w'
\]

\[
= \rho_3 \ (h \ (\phi_3 w) \ (\phi_3 w')) \ w'
\]

Therefore

\[
(D_W \circ (\phi_3 \circ \rho_3) \circ D_Z) \ h \ w
\]

\[
= \rho_3 \ (h \ (\phi_3 w) \ (\phi_3 w)) \ w
\]

\[
= (D_W \circ (\phi_3 \circ (D_W \circ (\phi_3 \circ \rho_3)))) \ h \ w,
\]
from which the desired equality follows. Now let's verify associativity:

\[
( (\phi_3, \rho_3) \circ (\phi_2, \rho_2) ) \circ (\phi_1, \rho_1)
\]

\[
= (\phi_2 \circ \phi_3, D_W \circ ( (\phi_3 \circ \rho_3) \circ \rho_2 ) \circ (\phi_1, \rho_1)
\]

\[
= (\phi_1 \circ \phi_2 \circ \phi_3, D_W \circ ( (\phi_3 \circ (D_W \circ (\phi_3 \circ \rho_3)) ) \circ (\phi_2 \circ \rho_2) \circ \rho_1 )
\]

since \( \circ \) is a functor,

\[
= (\phi_1 \circ \phi_2 \circ \phi_3, D_W \circ (\phi_3 \circ \rho_3) \circ D_Z \circ (\phi_2 \circ \rho_2) \circ \rho_1 )
\]

by the equality demonstrated earlier,

\[
= (\phi_3, \rho_3) \circ (\phi_1 \circ \phi_2, D_Z \circ (\phi_2 \circ \rho_2) \circ \rho_1 )
\]

\[
= (\phi_3, \rho_3) \circ ( (\phi_2, \rho_2) \circ (\phi_1, \rho_1 )
\]

as desired.

The identity morphisms for \( \Sigma \) are given by

\[
(1_X, I_X) \in X \xrightarrow{\Sigma} X,
\]

where \( 1_X \in X + X \) is the identity function and \( I_X \in X \rightarrow (X \rightarrow X) \) is given by \( I_X(x, x') = x \), for all \( x, x' \in X \). The computations needed to prove this are very easy.

The morphisms in \( \Sigma \) have a mysterious quality about them. The following theorem dispels some of the mystery. If one knows what "split epimorphisms" are in, say, the category of right modules over a ring, then this theorem can be interpreted as saying the arrows in \( \Sigma \) are "split
surjections" in Set.

**Theorem 6.1:** A morphism $<\phi, \rho> \in X \rightarrowtail_{\Sigma} Y$ induces an equivalence relation ~ on Y such that

$$(Y, \phi \in Y \rightarrow X, \pi \in Y \rightarrow Y/\sim)$$

is a product in Set of X and Y/~, where $\pi \in Y \rightarrow Y/\sim$ maps each element of Y to its equivalence class. In particular, $\phi$ is surjective.

**Proof:** Define ~ on Y by

$$y_1 \sim y_2 \iff \text{for all } x \in X, \rho \times y_1 = \rho \times y_2.$$

Then ~ is clearly an equivalence relation.

Consider in Set the diagram

```
      Z
     / \    /
   /   \  /   \
  \   /  /   \\
   \ /  /    \\
    | /\      \\
    | /  \     \\
    |   \    \\
    |    \   \\
    X<----------Y<---------Y/~
      \   /      \   /
       \ /       \ /
        \|        \\
          |\        \\
          |  \      \\
          |   \     \\
          |    \    \\
          |     \   \\
          |      \ \\
          |       \     (\phi \circ \gamma)z = \phi (\gamma z)
```

We must show there exists a unique function $\gamma$ making the diagram commute. Let $z \in Z$. Define $\gamma$ by

$$\gamma z = \rho (\alpha z) y$$

where $y \in Y$ is any element such that $\pi y = \beta z$;

that $\gamma$ is well-defined is immediate from the definition of ~. Compute:

$$(\phi \circ \gamma)z = \phi (\gamma z)
= \phi (\rho (\alpha z) y)
= \alpha z, \text{ by } (\Sigma 1),$$
\[(\pi \circ \gamma) \ z = \pi (\gamma \ z)\]
\[= \pi (\rho \ (\alpha \ z) \ y)\]
\[= \pi \ y\]

because, for all \(x \in X\),
\[\rho \ x \ (\rho \ (\alpha \ z) \ y) = \rho \ x \ y, \text{ by (S3)}.\]

Hence \(\gamma\) makes the diagram commute. Suppose also that

\[
\begin{array}{ccc}
  & Z & \\
\alpha & \downarrow & \beta \\
X & \phi & \gamma' \\
  & Y & \pi \\
  & \downarrow & \rightarrow \\
  & Y/\sim & \\
\end{array}
\]

commutes; we must show \(\gamma = \gamma'\). Again, let \(z \in Z\). Then
\(\gamma \ z \sim \gamma' \ z\) because
\[\pi (\gamma \ z) = \beta \ z = \pi (\gamma' \ z).\]

Therefore,
\[
\begin{align*}
\gamma \ z &= \rho (\phi (\gamma \ z)) (\gamma \ z), \text{ by (S2)}, \\
&= \rho (\alpha \ z) (\gamma \ z) \\
&= \rho (\alpha \ z) (\gamma' \ z), \text{ since } \gamma \ z \sim \gamma' \ z, \\
&= \rho (\phi (\gamma' \ z)) (\gamma' \ z) \\
&= \gamma' \ z, \text{ by (S2)}. \\
\end{align*}
\]

So \(\gamma = \gamma'\), and we are done. \(\Box\)
CHAPTER VII

DESUGARED ALGOL: DIRECT SEMANTICS

In this chapter we will use the methods developed earlier to describe the semantics of a desugared version of that part of an ALGOL-like language which can be described without continuations.

Underlying an ALGOL-like language is a first-order data type language. Thus, there is a poset $\mathcal{D}$ of data types, a collection $\text{Op}$ of operators, and two functions

$$\text{arity} \in \text{Op} \to (\text{Ob } \mathcal{D})^*, \quad \text{res} \in \text{Op} \to \text{Ob } \mathcal{D}.$$  

Thus arity gives a finite sequence of data types for each operator, thereby specifying the data types to which an operator may meaningfully be applied, and res gives the data type that is the result of a meaningful application. This information amounts to a syntactic specification of the first-order data type language.

We also have in mind a specific semantics for these entities. Thus, we assume we are given a functor

$$\text{Val} \in \mathcal{D} \to \text{Set}$$

and an appropriate interpretation for the operators in the form of $m_{\text{Op}}$, an Op-indexed collection of functions such that for each $g \in \text{Op}$,

$$m_{\text{Op}} g \in \text{Val } \delta_1 \times \cdots \times \text{Val } \delta_n \to \text{Val } \delta$$

where arity $g = (\delta_1, \ldots, \delta_n)$ and res $g = \delta$. 
For example, it may be that $\mathcal{D}$ is given by the Hasse diagram

\[
\begin{array}{c}
\text{real} \\
\downarrow \\
\text{int} \\
\end{array}
\begin{array}{c}
\text{bool}
\end{array}
\begin{array}{c}
\text{bool}
\end{array}
\begin{array}{c}
\text{bool}
\end{array}
\begin{array}{c}
\text{bool}
\end{array}
\]

and $\text{Op}$, arity, and res is given by the following table

<table>
<thead>
<tr>
<th>$g \in \text{Op}$</th>
<th>arity $g$</th>
<th>res $g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{true}$</td>
<td>$\langle \rangle$</td>
<td>bool</td>
</tr>
<tr>
<td>$\text{false}$</td>
<td>$\langle \rangle$</td>
<td>bool</td>
</tr>
<tr>
<td>$\text{and}$</td>
<td>$\langle \text{bool, bool} \rangle$</td>
<td>bool</td>
</tr>
<tr>
<td>$0$</td>
<td>$\langle \rangle$</td>
<td>int</td>
</tr>
<tr>
<td>$1$</td>
<td>$\langle \rangle$</td>
<td>int</td>
</tr>
<tr>
<td>$\pi$</td>
<td>$\langle \rangle$</td>
<td>real</td>
</tr>
<tr>
<td>$+$</td>
<td>$\langle \text{real, real} \rangle$</td>
<td>real</td>
</tr>
<tr>
<td>$\text{succ}$</td>
<td>int</td>
<td>int</td>
</tr>
</tbody>
</table>

It would then be reasonable for Val to take $\mathcal{D}$ to the diagram

\[
\begin{array}{c}
\mathbb{Z} \\
\uparrow \\
\text{inclusion}
\end{array}
\begin{array}{c}
\{\text{true, false}\}
\end{array}
\]

and for the interpretations of the operators,

\[
\begin{align*}
\mathcal{M}_{\text{Op}} \text{true} & \in \{\langle \rangle\} \rightarrow \{\text{true, false}\} \\
\mathcal{M}_{\text{Op}} \text{false} & \in \{\langle \rangle\} \rightarrow \{\text{true, false}\} \\
\mathcal{M}_{\text{Op}} \text{and} & \in \{\text{true, false}\} \times \{\text{true, false}\} \rightarrow \{\text{true, false}\} \\
\mathcal{M}_{\text{Op}} 0 & \in \{\langle \rangle\} \rightarrow \mathbb{Z} \\
\mathcal{M}_{\text{Op}} 1 & \in \{\langle \rangle\} \rightarrow \mathbb{Z} \\
\mathcal{M}_{\text{Op}} \pi & \in \{\langle \rangle\} \rightarrow \mathbb{R}
\end{align*}
\]
\[ m_{\text{Op}} + \in \mathbb{R} \times \mathbb{R} \times \mathbb{R} \]
\[ m_{\text{Op succ}} \in \mathbb{Z} \times \mathbb{Z}, \]
to be the obvious ones.

An important idea is that for each data type \( \delta \in D \), Val \( \delta \) is a set of storable values.

From \( D \) we get \( P \), the poset of primitive phrase types.

Let
\[
\text{Ob } P = \{\text{comm}\}
\]
\[ \cup \{\delta\text{-exp} \mid \delta \in D\} \]
\[ \cup \{\delta\text{-acc} \mid \delta \in D\} \]
\[ \cup \{\delta\text{-var} \mid \delta \in D\}. \]

The partial order on \( P \) is given by
\( \pi \leq P \pi' \) iff \( \pi = \pi' \),
or \( \pi = \delta\text{-exp}, \pi' = \delta'\text{-exp}, \) where \( \delta \leq P \delta' \),
or \( \pi = \delta'\text{-acc}, \pi' = \delta\text{-acc}, \) where \( \delta \leq P \delta' \),
or \( \pi = \delta\text{-var}, \pi' = \delta'\text{-exp}, \) where \( \delta \leq P \delta' \),
or \( \pi = \delta'\text{-var}, \pi' = \delta\text{-acc}, \) where \( \delta \leq P \delta' \).

Note that \text{comm} cannot be compared with the other primitive phrase types. For example, if \( D \) were given by
\[
\begin{align*}
\text{real} \\
\mid \\
\text{int} \\
\end{align*}
\]
then \( P \) would be
For more discussion, see Reynolds [8]. (However, unlike Reynolds [8], we do not permit fully generic operations. As a consequence, the \( \delta_1 \delta_2 \)-variable construction is not needed.)

Now \( P \) generates the free type algebra \( \mathcal{T} \). Recall that
\[
\text{Ob} \left| \mathcal{T} \right| \text{is the disjoint union}
\]
\[
\{ns\} \cup P \cup \{\tau \mapsto \emptyset \mid \tau, \emptyset \in \mathcal{T}\}.
\]
Once we are given \( \text{Id} \), an infinite set of identifiers, we have \( \mathcal{A} \), the poset of phrase type assignments, which played an important role in the preceding development.

We view the language \( L \) (desugared ALGOL) as a free \( \Lambda \)-algebra generated by an appropriate set of \( G \) of linguistic constants. Recall that we must also give
\[
\text{type} \in G \to \text{Ob} \mathcal{T},
\]
so that \( G \) is really a typed set of linguistic constants.

Take \( G \) to be the disjoint union of \( \text{Op} \) and \( \text{Op}' \) (to be defined shortly). For \( g \in \text{Op} \), let
\[
\text{type} \; g = \delta_1 - \text{exp} \to \delta_2 - \text{exp} \to \ldots \to \delta_n - \text{exp} \to \delta - \text{exp},
\]
where \( \text{arity} \; g = \langle \delta_1, \ldots, \delta_n \rangle \) and \( \text{res} \; g = \delta \). So, partly because product types are not in \( \mathcal{T} \), \( G \) contains curried versions of operators in \( \text{Op} \). As we will see, this works out very neatly. The typed set \( \text{Op}' \) is given by the following
table, where $\delta \in D$ is a data type and $\tau \in T$ is a phrase type.

\[
\begin{align*}
\text{g} & \in \text{Op'} & \text{type g} \\
\text{skip} & & \text{comm} \\
; & & \text{comm} = \text{comm} = \text{comm} \\
:= \delta & & \delta\text{-acc} \Rightarrow \delta\text{-exp} \Rightarrow \text{comm} \\
\text{ifthenelse}_\tau & & \text{bool-exp} \Rightarrow \tau \Rightarrow \tau \Rightarrow \tau \\
\text{rec}_\tau & & (\tau \Rightarrow \tau) \Rightarrow \tau \\
\text{newvar}_\delta & & (\delta\text{-var} \Rightarrow \text{comm}) \Rightarrow \text{comm}
\end{align*}
\]

It is the presence of Op' as a subset of G that entitles us to regard \( l \) as a desugared version of ALGOL. Intuitively, skip is the "do nothing" command, the operator ; concatenates commands, := is assignment for the data type $\delta$, ifthenelse is the conditional for the phrase type $\tau$, rec is used in the desugared denotation of recursively defined entities (e.g. \( \text{while } b \text{ do } c \)

\[\text{rec}_{\text{comm}} [\lambda x:\text{comm}. [\text{ifthenelse}_{\text{comm}} b [c;x] \text{ skip}]]]]\),
and newvar is used to desugar declarations of $\delta$-variables (e.g. 

\[\text{begin } \delta\text{-var } x ; c \text{ end} \]

is really

\[ [\text{newvar}_\delta [\lambda x:\delta\text{-var}. c]]].\]

To give the semantics of this language we must do three things. First, we must give a Cartesian closed category and its associated canonical type algebra $K$. Second, we must give a functor $\text{mg} \in P + | K |$, which by Theorem 2.2
can always be uniquely extended to a type algebra homomorphism \( \operatorname{mng} \in T \xrightarrow{\text{TypeAlg}} K \). Finally, we must for each generator \( g \in G \) give its semantics, i.e.

\[
\operatorname{semf} \ x \in \operatorname{Env} \xrightarrow{\text{empty}} \operatorname{mng} (\text{type } g).
\]

By the Fundamental Theorem of Semantics, this semantic function defined on generators uniquely determines a properly behaved semantic function defined on all program fragments.

Let \( \Sigma \) denote the category of store shapes described in Chapter VI. Let \( |K| \) be the full subcategory of \( \Sigma \to \text{Pdom} \) whose objects are functors \( F \in \Sigma \to \text{Pdom} \) such that

1. \( F S \) is a domain for every \( S \in \text{Ob } \Sigma \), and
2. \( F \langle \phi, \rho \rangle \) is a strict, continuous function for every \( \langle \phi, \rho \rangle \in \text{Ar } \Sigma \). In Chapter V we showed this category is Cartesian closed. Let \( K \) denote the corresponding type algebra. This takes care of the first, and simplest, part of giving the semantics of \( l \).

We next embark on the description of the meaning functor \( \operatorname{mng} \in \text{P} \to |K| \).

Therefore we must give functors

\[
\begin{align*}
\operatorname{mng} & \xrightarrow{\text{comm}} \\
\operatorname{mng} & \xrightarrow{\delta\text{-exp}} \\
\operatorname{mng} & \xrightarrow{\delta\text{-acc}} \\
\operatorname{mng} & \xrightarrow{\delta\text{-var}}
\end{align*}
\]

\( \xrightarrow{\Sigma \to \text{Pdom} \text{, where } \delta \in D} \)

and we must show these functors are actually objects of \( |K| \) by verifying conditions (1) and (2) above. Before proceeding, a definition is in order. If \( \rho \in X \to (Y \to Y) \), then
\[ \rho_\bot \in X_\bot \xrightarrow{P_{\text{dom}}} (Y \Rightarrow Y_\bot) \] is the strict extension of \( \rho \), i.e.

for \( x \in X_\bot \), \( y \in Y \),

\[
\rho_\bot \ x \ y = \begin{cases} 
\rho \ x \ y, & \text{if } x \neq \bot \\
\bot, & \text{if } x = \bot 
\end{cases}
\]

Intuitively, commands are possibly nonterminating transformations of the store. Thus, for \( X \in \text{Ob } \Sigma \), let

\[ \text{mng } \text{comm } X = X \Rightarrow X_\bot. \]

For \( (\phi, \rho) \in X \xrightarrow{\Sigma} Y \), let

\[ \text{mng } \text{comm } (\phi, \rho) \in (X \Rightarrow X_\bot) \xrightarrow{P_{\text{dom}}} (Y \Rightarrow Y_\bot) \]

be given by

\[ \text{mng } \text{comm } (\phi, \rho) \ c \ y = \rho_\bot \ (c \ (\phi \ y)) \ y, \]

where \( c \in X \xrightarrow{P_{\text{dom}}} X_\bot \) and \( y \in Y \). This last equation says that to view a command \( c \), defined for a "little" store, as a command for a "big" store, do the following:

1) forget some of the "big" store via \( \phi \),
2) apply \( c \),
3) overwrite the "big" store using the result of \( c \), providing \( c \) terminates.

Note that we could almost write

\[ \text{mng } \text{comm } (\phi, \rho) = D_Y \circ (\phi \circ \rho_\bot), \]

but the domains and codomains don’t quite match. We must check that \( \text{mng } \text{comm } (\phi, \rho) \) is continuous. Suppose \( S \) is a directed subset of \( X \Rightarrow X_\bot \). For all \( y \in Y \),
\[ \text{mng comm } (\phi, \rho) \ (\text{mng comm } X \ S) \ y \]
\[ = \rho \ (\text{mng comm } X \ S \ (\phi y)) \ y \]
\[ = \rho \ (\text{mng comm } \ S \ (\phi y)) \ y \]
\[ = \text{mng comm } \ Y \ (\rho \ (\text{mng comm } \ S \ (\phi y))) \ y, \]

since \( \rho \) is continuous,
\[ = \text{mng comm } \ Y \ (\rho \ (\text{mng comm } \ S \ (\phi y))) \ y \]
\[ = \text{mng comm } \ Y \ (\text{mng comm } (\phi, \rho) \ S) \ y \]
\[ = \text{mng comm } \ Y \ (\text{mng comm } (\phi, \rho) \ S) \ y. \]

Therefore
\[ \text{mng comm } (\phi, \rho) \ (\text{mng comm } X \ S) \]
\[ = \text{mng comm } \ Y \ (\text{mng comm } (\phi, \rho) \ S), \]

which implies \( \text{mng comm } (\phi, \rho) \) is continuous. Also, we must check that \( \text{mng comm } \) is truly a functor. It is trivial to verify that \( \text{mng comm } \) preserves identity morphisms. Suppose
\[ (\phi, \rho) \in X \rightarrow \ Y, \quad (\phi^{'}, \rho^{'}) \in Y \rightarrow Z. \]

We must establish
\[ \text{mng comm } ((\phi^{'}, \rho^{'}) \circ (\phi, \rho)) = (\text{mng comm } (\phi^{'}, \rho^{'}) \circ (\text{mng comm } (\phi, \rho))). \]

Let \( c \in X \rightarrow X^\perp \) and \( z \in Z. \) Then
\[ \text{mng comm } ((\phi^{'}, \rho^{'}) \circ (\phi, \rho)) \ c \ z \]
\[= \text{mng comm } (\phi \circ \phi', D_Z \circ (\phi' \circ \rho) \circ \rho) \text{ c z}\]

\[= (D_Z \circ (\phi' \circ \rho) \circ \rho) \perp (\text{c } (\phi (\phi' z))) \text{ z, otherwise},\]

\[= \begin{cases} 1, & \text{if } \text{c } (\phi (\phi' z)) = 1, \\ D_Z ((\phi' \circ \rho) \circ (\text{c } (\phi (\phi' z)))) \text{ z, otherwise}, \end{cases}\]

\[= \begin{cases} 1, & \text{if } \text{c } (\phi (\phi' z)) = 1, \\ ((\phi' \circ \rho) \circ (\text{c } (\phi (\phi' z)))) \text{ z z, otherwise}, \end{cases}\]

\[= \begin{cases} 1, & \text{if } \text{c } (\phi (\phi' z)) = 1, \\ (\rho' \circ (\text{c } (\phi (\phi' z)))) \circ \phi' \text{ z z, otherwise}, \end{cases}\]

\[= \begin{cases} 1, & \text{if } \text{c } (\phi (\phi' z)) = 1, \\ \rho' \circ (\text{c } (\phi (\phi' z))) \text{ (phi' z)} \text{ z, otherwise}, \end{cases}\]

\[= \rho' \perp (\text{mng comm } (\phi, \rho) \text{ c } (\phi' z)) \text{ z}\]

\[= \text{mng comm } (\phi', \rho' \perp (\text{mng comm } (\phi, \rho) \text{ c } (\phi' z)) \text{ z}\]

\[= (\text{mng comm } (\phi', \rho' \perp (\text{mng comm } (\phi, \rho) \text{ c } (\phi' z)) \text{ z, as desired.} \text{ A moment's thought shows that}\]

(1) for \(X \in \text{Ob } \Sigma\), \text{mng comm } X \text{ is a domain, and}\]

(2) for \((\phi, \rho) \in \text{Ar } \Sigma\), \text{mng comm } (\phi, \rho) \text{ is strict.}\]

Thus \text{mng comm} is an object of \(|K|\).

Suppose \(\delta \in \mathcal{D}\) is a data type. Our intuition about \(\delta\)-expressions is that they try to produce elements of \(\text{Val } \delta\) from the current store. Thus the functor \text{mng } \delta\text{-exp } \in \Sigma \to \text{Pdom}\) is defined on an object \(X\) of \(\Sigma\) by

\[
\text{mng } \delta\text{-exp } X = X \Rightarrow (\text{Val } \delta) \perp
\]

and on an arrow \((\phi, \rho) \in X \xrightarrow{\Sigma} Y\) by

\[
\text{mng } \delta\text{-exp } (\phi, \rho) \in (X \Rightarrow (\text{Val } \delta) \perp) \xrightarrow{\text{Pdom}} (Y \Rightarrow (\text{Val } \delta) \perp),
\]
and

\[ \text{mng} \, \delta\text{-exp} \, \langle \phi, \rho \rangle \, \varepsilon \, Y = \varepsilon \, (\phi y) , \]

where \( e \in X \xrightarrow{p_{\text{dom}}} (\text{Val} \, \delta)_1 \) and \( y \in Y \). Note that we use

\((\text{Val} \, \delta)_1\), rather than \(\text{Val} \, \delta\), because the evaluation of an expression does not necessarily terminate. It is clear that if \(\text{mng} \, \delta\text{-exp}\) is a functor, then it satisfies (1) and (2) and is therefore an object of \(|K|\). The neatest way to see that \(\text{mng} \, \delta\text{-exp}\) is indeed a functor is to consider the forgetful functor

\[ U \in \Sigma \xrightarrow{\text{Pdom}^{\text{op}}} , \]

given by

\[ UX = X \text{ for } X \in \text{Ob} \, \Sigma , \quad U(\phi, \rho) = \phi \text{ for } \langle \phi, \rho \rangle \in \text{Ar} \, \Sigma , \]

and then to notice that

\[ \text{mng} \, \delta\text{-exp} = (U -) = (\text{Val} \, \delta)_1 \]

where the heavy arrow is the internal hom functor for \(\text{Pdom}\).

Next, we think about \(\delta\text{-acceptors}\). They take elements of \(\text{Val} \, \delta\) (perhaps from the right-hand side of an assignment command) and produce transformations of the store. Therefore, for a store shape \(X \in \Sigma\), let

\[ \text{mng} \, \delta\text{-acc} X = \text{Val} \, \delta \to \text{mng} \, \text{comm} \, X , \]

and, for \(\langle \phi, \rho \rangle \in X \xrightarrow{\Sigma} Y\), let

\[ \text{mng} \, \delta\text{-acc} \, \langle \phi, \rho \rangle \, \varepsilon \, (\text{Val} \, \delta \to \text{mng} \, \text{comm} \, X) \xrightarrow{\text{Pdom}} (\text{Val} \, \delta \to \text{mng} \, \text{comm} \, Y) \]

be given by

\[ \text{mng} \, \delta\text{-acc} \, \langle \phi, \rho \rangle \, a \, v = \text{mng} \, \text{comm} \, \langle \phi, \rho \rangle \, (a \, v) . \]
If we rewrite this definition as

$$\text{mng } \delta\text{-acc } = \text{Val } \delta \Rightarrow (\text{mng } \text{comm } \cdot),$$

where the heavy arrow is the internal hom functor for $\mathcal{P}_\text{dom}$, it becomes clear that $\text{mng } \delta\text{-acc }$ is a functor. Once again, it is clear that $\text{mng } \delta\text{-acc }$ is an object of $|K|$. Finally, $\delta$-variables must have in them enough structure to be used as either $\delta$-acceptors or $\delta$-expressions. Therefore, let

$$\text{mng } \delta\text{-var } = \text{mng } \delta\text{-acc } \times \text{mng } \delta\text{-exp },$$

where the $\times$ is the distinguished binary product in $|K|$. In particular, notice that for a store shape $X \in \text{Ob } \Sigma$

$$\text{mng } \delta\text{-var } X = \text{mng } \delta\text{-acc } X \times \text{mng } \delta\text{-exp } X.$$  

So far, we have defined $\text{mng }$ on objects of $\mathcal{P}$. Next, for each $\pi \leq \mathcal{P} \pi'$, we must give a natural transformation

$$\text{mng } (\pi \leq \mathcal{P} \pi') \in \text{mng } \pi \rightarrow \text{mng } \pi'.$$

There are five cases, corresponding to the five kinds of implicit conversions which comprise the arrows of $\mathcal{P}$.

1. If $\pi = \pi'$, then let $\text{mng } (\pi \leq \mathcal{P} \pi)$ be the identity natural transformation on the functor $\text{mng } \pi$.

2. The construction

$$\text{mng } \delta\text{-exp } = (-) \Rightarrow (\text{Val } \delta) \downarrow$$

is functorial in $\delta$. Thus, if $\delta \leq \mathcal{P} \delta'$, then we obtain a natural transformation.

$$\text{mng } (\delta\text{-exp } \leq \delta'\text{-exp }) \in \text{mng } \delta\text{-exp } \rightarrow \text{mng } \delta'\text{-exp }$$

by letting

$$\text{mng } (\delta\text{-exp } \leq \delta'\text{exp }) = (-) \Rightarrow (\text{Val } (\delta \leq \mathcal{P} \delta')) \downarrow$$
Therefore, if $X \in \text{Ob } \Sigma$ and $e \in X \xrightarrow{\text{Pdom}} (\text{Val } \delta)_\bot$, then

$$\text{mng } (\delta\text{-exp} \leq \delta'\text{-exp}) X e \xrightarrow{(\text{Val } \delta)_\bot} X \xrightarrow{(\text{Val } \delta')_\bot} \text{Pdom}$$

and

$$\text{mng } (\delta\text{-exp} \leq \delta'\text{-exp}) X e = (\text{Val } (\delta \leq \delta'))_\bot \circ e.$$

(3) Another construction functorial in $\delta$ (but contravariant!) is

$$\text{mng } \delta\text{-acc} = \text{Val } \delta = (\text{mng } \text{comm } -);$$

hence, if $\delta \leq \delta'$, then we can get a natural transformation

$$\text{mng } (\delta'\text{-acc} \leq_p \delta\text{-acc}) \in \text{mng } \delta'\text{-acc} \xrightarrow{|K|} \text{mng } \delta\text{-acc}$$

by letting

$$\text{mng } (\delta'\text{-acc} \leq_p \delta\text{-acc}) = (\text{Val } (\delta \leq \delta'))_\bot \circ (\text{mng } \text{comm } -).$$

In this situation, if $X \in \text{Ob } \Sigma$ and $a \in \text{Val } \delta' \xrightarrow{\text{Pdom}} \text{mng } \text{comm } X$, then

$$\text{mng } (\delta'\text{-acc} \leq_p \delta\text{-acc}) X \xrightarrow{\text{mng } \delta'\text{-acc} X} \text{mng } \delta\text{-acc} X \xrightarrow{\text{Pdom}} \text{mng } \delta\text{-acc} X$$

and

$$\text{mng } (\delta'\text{-acc} \leq_p \delta\text{-acc}) X a = a \circ \text{Val } (\delta \leq \delta').$$

(4) If $\delta \leq \delta'$, then define

$$\text{mng } (\delta\text{-var} \leq_p \delta\text{-exp}) \in \text{mng } \delta\text{-var} \xrightarrow{|K|} \text{mng } \delta\text{-exp}$$

to be

$$\text{mng } (\delta\text{-var} \leq_p \delta\text{-exp}) = \text{mng } (\delta\text{-exp} \leq_p \delta'\text{-exp}) \circ \text{proj}_2$$

where

$$\text{proj}_2 \in \text{mng } \delta\text{-var} = \text{mng } \delta\text{-acc} \times \text{mng } \delta\text{-exp} \xrightarrow{|K|} \text{mng } \delta\text{-exp}$$

is the projection onto the second component.
(5) If \( \delta \leq \delta' \), then define

\[
\text{mng} \ (\delta'-\text{var} \leq_p \delta-\text{acc}) \in \text{mng} \delta'-\text{var} \xrightarrow{|K|} \text{mng} \delta-\text{acc}
\]

to be

\[
\text{mng} \ (\delta'-\text{var} \leq_p \delta-\text{acc}) = \text{mng} \ (\delta-\text{acc} \leq_p \delta-\text{acc}) \circ \text{proj}_1
\]

where

\[
\text{proj}_1 \circ \text{mng} \delta'-\text{var} = \text{mng} \delta'-\text{acc} \times \text{mng} \delta'-\exp \xrightarrow{|K|} \text{mng} \delta'-\text{acc}
\]

is the projection onto the first component. An easy argument shows that we really have defined a functor

\[
\text{mng} \in \mathcal{P} \to |K|.
\]

As we noted earlier, there is a unique type algebra homomorphism

\[
\text{Mng} \in T \to K
\]

that extends mng.

The rest of this chapter is devoted to giving the semantics of the generating set \( G \) of linguistic constants.

Let \( g \in \text{Op} \) with arity \( g = \{\delta_1, \ldots, \delta_n\} \), \( \text{res} \ g = \delta \).

We must define

\[
\text{semf} \ g \in \text{Env} \circ \text{emp}_T \xrightarrow{|K|} (\text{Mng} \delta_1-\exp \Rightarrow \ldots \Rightarrow \text{Mng} \delta_n-\exp, \text{Mng} \delta-\exp).
\]

The codomain of this natural transformation is a rather formidable functor. However, since \(|K|\) is a Cartesian closed category, there is a canonical bijection between

\[
\text{Env} \circ \text{emp}_T \xrightarrow{|K|} (\text{Mng} \delta_1-\exp \Rightarrow \ldots \Rightarrow \text{Mng} \delta_n-\exp, \text{Mng} \delta-\exp)
\]

and
\[(\text{Mng } \delta_1^{\text{exp}} \times \ldots \times \text{Mng } \delta_n^{\text{exp}}) \to \text{Mng } \delta^{\text{exp}}]_K\]

based on repeated abstraction and the fact that the product with a terminal object \((\text{Env } \text{emp}_T \text{ in this case})\) is naturally isomorphic to the identity. So it suffices to give the corresponding \(\hat{g} \in (\text{Mng } \delta_1^{\text{exp}} \times \ldots \times \text{Mng } \delta_n^{\text{exp}}) \to \text{Mng } \delta^{\text{exp}}.\)

For \(X \in \text{Ob } \Sigma,\) let
\[\hat{g} X \in \text{Mng } \delta_1^{\text{exp}} X \times \ldots \times \text{Mng } \delta_n^{\text{exp}} X \to \text{Mng } \delta^{\text{exp}} X\]
be given by
\[
\hat{g} X (e_1, \ldots, e_n) = \begin{cases} 
\text{m}_{\text{Op}} g (e_1 x, \ldots, e_n x) & \text{if, for all } i, e_i x \neq 1 \\
1 & \text{if, for some } i, e_i x = 1,
\end{cases}
\]

where \(x \in X\) and, for each \(i, \; e_i : X \to (\text{Val } \delta_i)_1.\)

It is easy to see that \(\hat{g} X\) is continuous. We must verify that \(\hat{g}\) is a natural transformation. To this end, suppose
\[(\phi, \rho) \in X \to Y,\] and consider the diagram
\[
\begin{align*}
\text{Mng } \delta_1^{\text{exp}} X \times \ldots \times \text{Mng } \delta_n^{\text{exp}} X & \to \text{Mng } \delta^{\text{exp}} X \\
\text{Mng } \delta_1^{\text{exp}} (\phi, \rho) \times \ldots \times \text{Mng } \delta_n^{\text{exp}} (\phi, \rho) \downarrow & \downarrow \text{Mng } \delta^{\text{exp}} (\phi, \rho) \\
\text{Mng } \delta_1^{\text{exp}} Y \times \ldots \times \text{Mng } \delta_n^{\text{exp}} Y & \to \text{Mng } \delta^{\text{exp}} Y.
\end{align*}
\]

Let \(e_i : X \to (\text{Val } \delta_i)_1\) for \(i = 1, \ldots, n,\) and let \(y \in Y.\)

Then
\[
(\hat{g} Y \circ (\text{Mng } \delta_1^{\text{exp}} (\phi, \rho) \times \ldots \times \text{Mng } \delta_n^{\text{exp}} (\phi, \rho))) (e_1, \ldots, e_n) Y
\]
\[ \hat{g} Y \langle \text{Mng} \delta_1 \text{-exp} (\phi, \rho), e_1, \ldots, \text{Mng} \delta_n \text{-exp} (\phi, \rho), e_n \rangle Y \]

\[ = \begin{cases} 
\hat{g} \langle e_1 (\phi y), \ldots, e_n (\phi y) \rangle & \text{if all } e_i (\phi y) \neq 1 \\
1 & \text{if some } e_i (\phi y) = 1 
\end{cases} \]

\[ = \hat{g} X \langle e_1, \ldots, e_n (\phi y) \rangle \]

\[ = \text{Mng} \delta \text{-exp} (\phi, \rho) \langle \hat{g} X \langle e_1, \ldots, e_n \rangle \rangle y \]

\[ = \left( \text{Mng} \delta \text{-exp} (\phi, \rho) \circ \hat{g} X \right) \langle e_1, \ldots, e_n \rangle y, \]

as desired.

Consider the "do nothing" command skip. We must give

\[ \text{semf skip } \in \text{Env emp}_T \xrightarrow{\text{skip}} \text{Mng comm}. \]

Let \( X \in \text{Ob } E \). Recall that \( \text{Env emp}_T X \) is the one-point predomain \( \{()\} \) and \( \text{Mng comm } X = X = X_\bot \). Then

\[ \text{semf skip } X \in \text{Env emp}_T X \longrightarrow \text{Mng comm } X \]

is given by

\[ \text{semf skip } X (\text{skip } x) = x, \text{ for all } x \in X. \]

Clearly, then, skip does nothing to the current store \( x \).

Obviously, \( \text{semf skip } X \) is continuous. To see that \( \text{semf skip } \) is a natural transformation, let \( \langle \phi, \rho \rangle \in X \longrightarrow Y \). In \( \text{Pdom} \)

consider the diagram

\[
\begin{array}{ccc}
\text{Env emp}_T X & \xrightarrow{\text{semf skip } X} & \text{Mng comm } X \\
\Downarrow & & \Downarrow \\
\text{Env emp}_T Y & \xrightarrow{\text{semf skip } Y} & \text{Mng comm } Y.
\end{array}
\]

We compute, for \( y \in Y \),
\[(\text{Mng comm} \ (\phi, \rho) \ \circ \ \text{semf} \ \text{skip} \ X) \ (\epsilon) \ y \]
\[= \text{Mng comm} \ (\phi, \rho) \ (\text{semf} \ \text{skip} \ X \ (\epsilon)) \ y \]
\[= \rho_\perp \ (\text{semf} \ \text{skip} \ X \ (\epsilon) \ (\phi \ y)) \ y \]
\[= \rho_\perp \ (\phi \ y) \ y \]
\[= y.
\]
The last equality follows from condition (22) in the definition of the category \(\Sigma\) of store shapes.

We wish to give
\[
\text{semf} \ \in \ \text{Env} \ \text{emp}_T \longrightarrow \text{Mng comm} = \text{Mng comm} \Rightarrow \text{Mng comm}, \quad |K| \]

Using the Cartesian closed category structure of \(|K|\), it suffices to define the corresponding natural transformation
\[
\text{sc} \ \in \ \text{Mng comm} \times \text{Mng comm} \longrightarrow \text{Mng comm} \quad |K| \]

For each store shape \(X \in \text{Ob} \ \Sigma\), we must give
\[
\text{sc} \ X \ \in \ \text{Mng comm} \ X \times \text{Mng comm} \ X \longrightarrow \text{Mng comm} \ X \quad Pdom
\]

Recall that
\[
\text{Mng comm} \ X = X \Rightarrow X_\perp. \ \text{For} \ f \ \in \ X \longrightarrow X_\perp, \ \text{let} \ f_\perp \ \in \ X \perp \ Pdom
\]
be the strict extension of \(f\). Now let
\[
\text{sc} \ X \ (c, \ c') = c' \perp \circ c,
\]
where \(c, \ c' \ \in \ X \longrightarrow X_\perp\). Note the order in which the commands are composed; it is determined by the operational rule that says \((c; c')\) means execute \(c\) and then execute \(c'\). (Of course, this is direct, not continuation, semantics.) Continuity of \(\text{sc} \ X\) follows from the continuity of composition. We must check that \(\text{sc} \) is indeed a natural transformation. Suppose \((\phi, \rho) \ \in \ X \longrightarrow Y\). We wish to show that
commutes. Let \( c, c' \in \text{Mng comm} \ X \) and \( y \in Y \). Then

\[
(sc \ Y \circ (\text{Mng comm} \ (\phi,\rho) \times \text{Mng comm} \ (\phi,\rho))) \ (c, c') \ y
\]

\[
= sc \ Y \ (\text{Mng comm} \ (\phi,\rho) \ c, \text{Mng comm} \ (\phi,\rho) \ c') \ y
\]

\[
= (\text{Mng comm} \ (\phi,\rho) \ c') \ (\text{Mng comm} \ (\phi,\rho) \ c \ y)
\]

\[
= (\text{Mng comm} \ (\phi,\rho) \ c') \ (\rho \ (c (\phi y)) \ y)
\]

\[
= \begin{cases} 
1, & \text{if } c (\phi y) = 1 \\
(\text{Mng comm} \ (\phi,\rho) \ c') \ (\rho \ (c (\phi y)) \ y), & \text{otherwise}
\end{cases}
\]

\[
= \begin{cases} 
1, & \text{if } c (\phi y) = 1 \\
\rho \ (c (\phi y)) \ (\rho \ (c (\phi y)) \ y), & \text{otherwise}
\end{cases}
\]

\[
= \begin{cases} 
1, & \text{if } c (\phi y) = 1 \text{ or } c' (c(\phi y)) = 1 \\
\rho \ (c (\phi y)) \ (\rho \ (c (\phi y)) \ y), & \text{otherwise}
\end{cases}
\]

\[
= \rho \ (c (\phi y)) \ y, \text{ otherwise, by (3)}
\]

\[
= \rho \ (sc \ X (c, c') (\phi y)) \ y
\]

\[
= \text{Mng comm} \ (\phi,\rho) \ (sc \ X (c, c')) \ y
\]

\[
= (\text{Mng comm} \ (\phi,\rho) \circ sc \ X) \ (c, c') \ y,
\]

which demonstrates commutativity. Note the usage of the fact that \( (\phi,\rho) \) satisfies (1) and (3).
Next we desire to define, for a data type $\delta$,
\[ \text{semf} := \delta \in \text{Env} \quad \text{emp} \quad \text{Mng} \quad \delta-\text{acc} \quad \text{Mng} \quad \delta-\text{exp} \quad \text{Mng} \quad \text{comm} \quad |K| \]

Because of the canonical bijection between
\[ \text{Env} \quad \text{emp} \quad \text{Mng} \quad \delta-\text{acc} \quad \text{Mng} \quad \delta-\text{exp} \quad \text{Mng} \quad \text{comm} \quad |K| \]

and
\[ \text{Mng} \quad \delta-\text{acc} \times \text{Mng} \quad \delta-\text{exp} \quad \text{Mng} \quad \text{comm}, \quad |K| \]

it is adequate to define the corresponding natural transformation
\[ \text{assign}_\delta \in \text{Mng} \quad \delta-\text{acc} \times \text{Mng} \quad \delta-\text{exp} \quad \text{Mng} \quad \text{comm} \quad |K| \]

Suppose $X \in \text{Ob} \ \Sigma$. We must give
\[ \text{assign}_\delta \ X \in \text{Mng} \quad \delta-\text{acc} \times \text{Mng} \quad \delta-\text{exp} \quad X \quad \text{Mng} \quad \text{comm} \quad X. \quad P_{\text{dom}} \]

Recall
\[ \text{Mng} \quad \delta-\text{acc} \quad X = \text{Val} \quad \delta \quad \text{Mng} \quad \text{comm} \quad X, \]
\[ \text{Mng} \quad \delta-\text{exp} \quad X = X \quad (\text{Val} \quad \delta)_1', \]
\[ \text{Mng} \quad \text{comm} \quad X = X \quad X_1. \]

For $a \in \text{Val} \quad \delta \quad \text{Mng} \quad \text{comm} \quad X, \ e \in X \quad (\text{Val} \quad \delta)_1', \ P_{\text{dom}}$

and $x \in X$, let
\[ \text{assign}_\delta \ X \quad (a, \ e) \quad x = a_1 \ (e \ x) \ x, \]
where $a_1$ is the strict extension of $a$. This definition reflects the idea that $(a \ :=_\delta \ e)$ means get a value from the store with $e$, then alter that same store with $a$.

The proof that $\text{assign}_\delta \ X$ is continuous is straightforward.

We would like to check that $\text{assign}_\delta$ is a natural transformation.

Let $(\phi, \rho) \in X \quad \rightarrow \ Y$. The diagram whose commutativity

we must verify is
\[ \text{Mng } \delta-\text{acc } X \times \text{Mng } \delta-\text{exp } X \xrightarrow{\text{assign}_\delta X} \text{Mng } \text{comm } X \]
\[ \text{Mng } \delta-\text{acc } (\phi, \rho) \times \text{Mng } \delta-\text{exp } (\phi, \rho) \]
\[ \xrightarrow{\text{assign}_\delta Y} \text{Mng } \text{comm } (\phi, \rho) \]
\[ \text{Mng } \delta-\text{acc } Y \times \text{Mng } \delta-\text{exp } Y \xrightarrow{\text{assign}_\delta Y} \text{Mng } \text{comm } Y \]

Let \( a \in \text{Val } \delta \xrightarrow{Pdom} \text{Mng } \text{comm } X, \ e \in X \xrightarrow{Pdom} (\text{Val } \delta)_\perp \),

and \( y \in Y \). Then

\[
\text{assign}_\delta Y \circ (\text{Mng } \delta-\text{acc } (\phi, \rho) \times \text{Mng } \delta-\text{exp } (\phi, \rho)) \langle a, e \rangle y
\]

\[ = \text{assign}_\delta Y (\text{Mng } \delta-\text{acc } (\phi, \rho) a, \text{Mng } \delta-\text{exp } (\phi, \rho) e) y
\]
\[ = (\text{Mng } \delta-\text{acc } (\phi, \rho) a)_\perp (\text{Mng } \delta-\text{exp } (\phi, \rho) e) y
\]
\[ = (\text{Mng } \delta-\text{acc } (\phi, \rho) a)_\perp (e (\phi) y) y
\]
\[ = \begin{cases} 
\text{Mng comm } (\phi, \rho) (a (e (\phi) y)) y & \text{if } e (\phi) y \neq 1 \\
1 & \text{if } e (\phi) y = 1 
\end{cases}
\]

\[ = \rho_\perp (a_\perp (e (\phi) y)) (\phi y) y 
\]
\[ = \rho_\perp (\text{assign}_\delta X (a, e) (\phi y)) y 
\]
\[ = \text{Mng comm } (\phi, \rho) (\text{assign}_\delta X (a, e)) y 
\]
\[ = (\text{Mng comm } (\phi, \rho) \circ \text{assign}_\delta X) (a, e) y, 
\]

so that the diagram commutes.

The next generator on the list is the conditional for type \( \tau \); thus we seek

\[ \text{semf } \text{ifthenelse } \tau \in \text{Env } \text{emp}_\tau \xrightarrow{Mng \ \text{bool-exp} = \text{Mng } \tau = \text{Mng } \tau = \text{Mng } \tau} \]

and it suffices to give the corresponding natural transformation in

\[ \text{Mng } \text{bool-exp} \times \text{Mng } \tau \times \text{Mng } \tau \xrightarrow{\text{K}} \text{Mng } \tau. \]
To do this will require some preliminary work.

Let the functor $B : T \to |K|$ be the product in the functor category $T \to |K|$ of the constant functor $\tau \mapsto \text{Mng bool-exp}$, $\text{Mng}$, and $\text{Mng}$.

Thus for $\tau \in T$

$$B \tau = \text{Mng bool-exp} \times \text{Mng} \tau \times \text{Mng} \tau$$

Shortly we will define a functor $N : T \to (|\Sigma| \to \text{Pdom})$.

Our plan is to define natural transformations

$$\Psi \tau : \varepsilon \circ B \tau \to N \tau,$$

$$\Delta \tau : \varepsilon N \tau \to \text{Mng} \tau$$

and to then obtain the desired element of $B \tau \to \text{Mng} \tau$ by composition:

$$\Delta \tau \circ \Psi \tau : \varepsilon \circ B \tau \to \text{Mng} \tau.$$

Our immediate aim is the definition of

$$N : T \to (|\Sigma| \to \text{Pdom})$$

as a composition of three functors. Note that the forgetful functor $U \in \Sigma \to \text{Pdom}^{\text{op}}$, which forgets the second component of morphisms, induces a functor

$$\langle U, \to \rangle : |\Sigma| \to (|\Sigma| \to (\text{Pdom}^{\text{op}} \times \text{Pdom}))$$

such that for $F \in \text{Ob} |\Sigma| \subseteq \Sigma \to \text{Pdom}$ and $X \in \text{Ob} \Sigma$

$$\langle U, \to \rangle F X = \langle U, F \rangle X = \langle UX, FX \rangle = \langle X, FX \rangle.$$

Next, the internal hom functor

$$\text{Pdom}_\to \in \text{Pdom}^{\text{op}} \times \text{Pdom} + \text{Pdom}$$

induces

$$(\text{Pdom}_\to)^{\Sigma} \in (|\Sigma| \to (\text{Pdom}^{\text{op}} \times \text{Pdom})) \to (|\Sigma| \to \text{Pdom}),$$
which is such that
\[(P\text{dom}_{\omega})^{\Sigma} G X = P\text{dom}_{\omega} (G X),\]
where \(G \in \Sigma \Rightarrow (P\text{dom}_{\omega}^{\text{op}} \times P\text{dom})\) and \(X \in \text{Ob} \Sigma\). Now let \(N\) be the composition
\[
\begin{align*}
\tau &\xrightarrow{Mng} |K| \xrightarrow{(U, -)} (\Sigma \Rightarrow (P\text{dom}_{\omega}^{\text{op}} \times P\text{dom})) \xrightarrow{(P\text{dom}_{\omega})^{\Sigma}} (\Sigma \Rightarrow P\text{dom}).
\end{align*}
\]
Thus, if \(\tau \in \tau\) and \(X \in \text{Ob} \Sigma\), then
\[
N \tau X = (P\text{dom}_{\omega})^{\Sigma} ((U, -) (Mng \tau)) X
= (P\text{dom}_{\omega})^{\Sigma} (U, Mng \tau) X
= P\text{dom}_{\omega} (UX, Mng \tau X)
= X \xrightarrow{\text{Mng} \tau X; P\text{dom}}
\]
therefore \(N \tau X\) is the collection of functions from the set of stores of shape \(X\) to the collection of conventional meanings of type \(\tau\) for stores of shape \(X\).

Next we will define for each store shape \(X \in \text{Ob} \Sigma\) the "choice function"
\[
\begin{align*}
\psi_{\tau X} \in \mathcal{B} \tau X &\xrightarrow{P\text{dom}} N \tau X \\
= (X \Rightarrow (\text{Val bool}_1)) \times \text{Mng} \tau X \times \text{Mng} \tau X &\xrightarrow{P\text{dom}} X \Rightarrow \text{Mng} \tau X
\end{align*}
\]
by
\[
\psi_{\tau X} (b, t_1, t_2) X = \begin{cases} 
  t_1 \text{ if } b \times = \text{true}, \\
  t_2 \text{ if } b \times = \text{false}, \\
  \text{Mng} \tau X \text{ if } b \times = 1,
\end{cases}
\]
where \(b \in X \xrightarrow{P\text{dom}} (\text{Val bool}_1)\) and \(t_1, t_2 \in \text{Mng} \tau X\), and \(X \in X\).

Here we have used the fact that \(\text{Mng} \tau X\) is actually a domain, and therefore it has a minimal element \(\text{Mng} \tau X^1\).
It takes only a moment's thought to see that \( \Psi \tau X \) is indeed continuous. We claim that \( \Psi \tau \) is a natural transformation:

\[
\Psi \tau \in B \tau \xrightarrow{\Sigma} N \tau. \quad \Sigma \Rightarrow P_{\text{dom}}
\]

Suppose \( (\phi, \rho) \in X \xrightarrow{\Sigma} Y \). We must check the commutativity of the diagram

\[
(X \Rightarrow (\text{Val boolean} \downarrow)) \times \text{Mng} \tau X \times \text{Mng} \tau X \xrightarrow{\Psi \tau X} X \Rightarrow \text{Mng} \tau X
\]

\[
(\phi \Rightarrow \text{Mng} \tau (\phi, \rho)) \times \text{Mng} \tau (\phi, \rho) \xrightarrow{\Psi \tau} \text{Mng} \tau (\phi, \rho)
\]

\[
(Y \Rightarrow (\text{Val boolean} \downarrow)) \times \text{Mng} \tau Y \times \text{Mng} \tau Y \xrightarrow{\Psi \tau Y} Y \Rightarrow \text{Mng} \tau Y
\]

Take \( b \in X \xrightarrow{\Sigma} (\text{Val boolean} \downarrow), t_1 \in \text{Mng} \tau X, t_2 \in \text{Mng} \tau X, \)

\( y \in Y \). Then

\[
(\Psi \tau Y \circ ((\phi \Rightarrow \text{Mng} \tau (\phi, \rho)) \times \text{Mng} \tau (\phi, \rho))) (b, t_1, t_2) y
\]

\[
= \Psi \tau Y (b \circ \phi, \text{Mng} \tau (\phi, \rho) t_1, \text{Mng} \tau (\phi, \rho) t_2) y
\]

\[
\begin{cases}
\text{Mng} \tau (\phi, \rho) t_1 & \text{if } b(\phi y) = \text{true}, \\
\text{Mng} \tau (\phi, \rho) t_2 & \text{if } b(\phi y) = \text{false}, \\
\text{Mng} \tau Y & \text{if } b(\phi y) = 1,
\end{cases}
\]

\[
= \begin{cases}
\text{Mng} \tau (\phi, \rho) t_1 & \text{if } b(\phi y) = \text{true}, \\
\text{Mng} \tau (\phi, \rho) t_2 & \text{if } b(\phi y) = \text{false}, \\
\text{Mng} \tau (\phi, \rho) \text{Mng} \tau Y & \text{if } b(\phi y) = 1,
\end{cases}
\]

because \( \text{Mng} \tau (\phi, \rho) \) is strict,

\[
= \text{Mng} \tau (\phi, \rho) \circ \Psi \tau X (b, t_1, t_2) (\phi y)
\]

\[
= (\phi \Rightarrow \text{Mng} \tau (\phi, \rho)) \circ \Psi \tau X (b, t_1, t_2) y
\]

\[
= ((\phi \Rightarrow \text{Mng} \tau (\phi, \rho)) \circ \Psi \tau X)(b, t_1, t_2) y.
\]

This proves the claim of naturality.

Finally we will define for each \( \tau \in T \) a "generalized diagonalization"
The method of definition is by induction on the structure of $\tau \in T$ (There should be a better way.) Hence we will give

1. $\Delta \pi$ for each primitive phrase type $\pi$,
2. $\Delta \text{ns}$
3. $\Delta (\tau \Rightarrow \theta)$, assuming $\Delta \tau$ and $\Delta \theta$ are known.

Case la: $\pi = \text{comm}$. For $X \in \text{Ob } \Sigma$, let

$\Delta \text{comm } X \in (X \Rightarrow \text{Mng comm } X) \xrightarrow{P\text{dom}} \text{Mng comm } X$

be given by

$\Delta \text{comm } X \cdot h \cdot x = h \cdot x \cdot x,$

where $h \in X \xrightarrow{P\text{dom}} \text{Mng comm } X$ and $x \in X$.

It is very easy to see that $\Delta \text{comm } X$ is continuous. To check naturality of $\Delta \text{comm}$, let $(\phi, \rho) \in X \xrightarrow{\Sigma} Y$.

Consider the diagram

Let $h \in X \xrightarrow{P\text{dom}} \text{Mng comm } X, y \in Y$. Then

$(\Delta \text{comm } Y \circ (\phi \Rightarrow \text{Mng comm } (\phi, \rho))) \cdot h \cdot y$

$= \Delta \text{comm } Y \cdot (\text{Mng comm } (\phi, \rho) \circ h \circ \phi) \cdot y$

$= (\text{Mng comm } (\phi, \rho) \circ h \circ \phi) \cdot y \cdot y$

$= \text{Mng comm } (\phi, \rho) \cdot (h \cdot (\phi \cdot y)) \cdot y$
\[ \rho_\perp (h (\phi \, y) (\phi \, y)) \, y \]
\[ = \rho_\perp (\Delta \, \text{comm} \, X \, h \, (\phi \, y)) \, y \]
\[ = \text{Mng} \, \text{comm} \, (\phi, \rho) \, (\Delta \, \text{comm} \, X \, h) \, y \]
\[ = (\text{Mng} \, \text{comm} \, (\phi, \rho) \circ \Delta \, \text{comm} \, X) \, h \, y, \]

which shows \( \Delta \, \text{comm} \) is natural.

**Case 1b:** \( \pi = \delta\text{-exp} \), where \( \delta \) is a data type. For a store shape \( X \in \text{Ob} \, \Sigma \), let

\[ \Delta \, \delta\text{-exp} \, X \in (X \Rightarrow \text{Mng} \, \delta\text{-exp} \, X) \xrightarrow{P\text{dom}} \text{Mng} \, \delta\text{-exp} \, X \]

be given by

\[ \Delta \, \delta\text{-exp} \, X \, h \, x = h \, x \, x \]

where \( h \in X \xrightarrow{P\text{dom}} \text{Mng} \, \delta\text{-exp} \, X \) and \( x \in X \). Proofs of continuity and naturality are essentially the same as for Case 1a.

**Case 1c:** \( \pi = \delta\text{-acc} \) where \( \delta \) is a data type. Let \( X \in \text{Ob} \, \Sigma \) be a store shape and let

\[ \Delta \, \delta\text{-acc} \, X \in (X \Rightarrow \text{Mng} \, \delta\text{-acc} \, X) \xrightarrow{P\text{dom}} \text{Mng} \, \delta\text{-acc} \, X \]

be given by

\[ \Delta \, \delta\text{-acc} \, X \, h \, v \, x = h \, x \, v \, x \]

where \( h \in X \xrightarrow{P\text{dom}} \text{Mng} \, \delta\text{-acc} \, X = X \xrightarrow{P\text{dom}} (\text{Val} \, \delta \Rightarrow \text{Mng} \, \text{comm} \, X), v \in \text{Val} \, \delta \), and \( x \in X \). It is routine to check continuity of \( \Delta \, \delta\text{-acc} \, X \). To prove that \( \Delta \, \delta\text{-acc} \) is a natural transformation, let \( (\phi, \rho) \in X \xrightarrow{\Sigma} Y \), and consider the diagram
Let \( h \in X \xrightarrow{P_{\text{dom}}} \text{Mng } \delta\text{-acc } X \) for all \( v \in \text{Val } \delta \), and \( y \in Y \).

Then

\[
(\Delta \delta\text{-acc } Y \circ (\phi \Rightarrow \text{Mng } \delta\text{-acc } (\phi, \rho))) \, h \, v \, y
\]

\[
= \Delta \delta\text{-acc } Y \, (\text{Mng } \delta\text{-acc } (\phi, \rho) \circ h \circ \phi) \, v \, y
\]

\[
= (\text{Mng } \delta\text{-acc } (\phi, \rho) \circ h \circ \phi) \, y \, v \, y
\]

\[
= \text{Mng } \delta\text{-acc } (\phi, \rho) \, (h \, (\phi \, y)) \, v \, y
\]

\[
= \text{Mng } \text{comm } (\phi, \rho) \, (h \, (\phi \, y)) \, v \, y
\]

\[
= \rho_{\bot} \, (h \, (\phi \, y) \, v \, (\phi \, y)) \, y
\]

\[
= \rho_{\bot} \, (\Delta \delta\text{-acc } X \, h \, v \, (\phi \, y)) \, y
\]

\[
= \text{Mng } \text{comm } (\phi, \rho) \, (\Delta \delta\text{-acc } X \, h \, v) \, y
\]

\[
= \text{Mng } \delta\text{-acc } (\phi, \rho) \, (\Delta \delta\text{-acc } X \, h) \, v \, y
\]

\[
= (\text{Mng } \delta\text{-acc } (\phi, \rho) \circ \Delta \delta\text{-acc } X) \, h \, v \, y
\]

which proves that the diagram commutes.

**Case 1d:** \( \pi = \delta\text{-var} \) where \( \delta \) is a data type. Since

\[
\text{Mng } \delta\text{-var} = \text{Mng } \delta\text{-acc } \times \text{Mng } \delta\text{-exp},
\]

it would be nice to give

\[
\Delta \delta\text{-var } \in N \delta\text{-var } \xrightarrow{\Sigma = P_{\text{dom}}} \text{Mng } \delta\text{-var }
\]

in terms of

\[
\Delta \delta\text{-acc } \in N \delta\text{-acc } \xrightarrow{\Sigma = P_{\text{dom}}} \text{Mng } \delta\text{-acc }
\]

and

\[
\Delta \delta\text{-exp } \in N \delta\text{-exp } \xrightarrow{\Sigma = P_{\text{dom}}} \text{Mng } \delta\text{-exp }.
\]

There is a natural isomorphism
\[ \eta \in N \delta \text{-} \text{var} \longrightarrow N \delta \text{-} \text{acc} \times N \delta \text{-} \text{exp} \]
\[ \Sigma \rightarrow P\text{dom} \]

where for \( X \in \text{Ob} \Sigma \)

\[ \eta X \in X \Rightarrow (\text{Mng} \ \delta \text{-} \text{acc} \ X \times \text{Mng} \ \delta \text{-} \text{exp} \ X) \]

\[ \boxed{P\text{dom}} \rightarrow (X \Rightarrow \text{Mng} \ \delta \text{-} \text{acc} \ X) \times (X \Rightarrow \text{Mng} \ \delta \text{-} \text{exp} \ X) \]

is the obvious bijection. Thus, we may take \( \Delta \ \delta \text{-} \text{var} \) to be the composite

\[ \Delta \ \delta \text{-} \text{var} = (\Delta \ \delta \text{-} \text{acc} \times \Delta \ \delta \text{-} \text{exp}) \circ \eta, \]

where the composition takes place in \( \Sigma \rightarrow P\text{dom} \).

Case 2: We want to define

\[ \boxed{\Delta \ \text{ns} \in N \ \text{ns} \longrightarrow \text{Mng} \ \text{ns} \quad \Sigma \rightarrow P\text{dom}} \]

There is only one way because \( \text{Mng} \ \text{ns} \) is a terminal object in \( \Sigma \rightarrow P\text{dom} \).

Case 3: Assume that

\[ \boxed{\Delta \ \tau \in N \ \tau \longrightarrow \text{Mng} \ \tau \quad \Sigma \rightarrow P\text{dom} \quad \Delta \ \theta \in N \ \theta \longrightarrow \text{Mng} \ \theta \quad \Sigma \rightarrow P\text{dom}} \]

are known. We will define

\[ \Delta \ (\tau \Rightarrow \theta) \in N \ (\tau \Rightarrow \theta) \longrightarrow \text{Mng} \ (\tau \Rightarrow \theta). \quad \Sigma \rightarrow P\text{dom} \]

(Actually, \( \Delta \ \tau \) will not enter into the definition of \( \Delta \ (\tau \Rightarrow \theta) \). This is a curious fact, and it should be connected with some interesting intuition.) We start with

\[ \text{Ap} = \text{Ap} \mid_K \left( \text{Mng} \ \tau, \ \text{Mng} \ \theta \right) \in \text{Mng} \ (\tau \Rightarrow \theta) \times \text{Mng} \ \tau \longrightarrow \text{Mng} \ \theta. \]

Hence, for \( X \in \text{Ob} \Sigma \),

\[ \text{Ap} \ X \in \text{Mng} \ (\tau \Rightarrow \theta) \ X \times \text{Mng} \ \tau \ X \longrightarrow \text{Mng} \ \theta \ X. \]
From Ap we get a natural transformation

\[ \Phi (\tau, \theta) \in N (\tau \to \theta) \times \text{Mng } \tau \to \text{N } \theta \]

where for \( X \in \text{Ob } \Sigma \)

\[ \Phi (\tau, \theta) X \in (X \to \text{Mng } (\tau \to \theta) X) \times \text{Mng } \tau X \to (X \to \text{Mng } \theta X) \]

is given by

\[ \Phi (\tau, \theta) X (h, t) x = \text{Ap } X (h x, t), \]

where \( h \in X \to \text{Mng } (\tau \to \theta) X, t \in \text{Mng } \tau X, \) and \( x \in X. \)

(The idea is this: a function like Ap X, say \( a \in (U \times T) \to V, \)

induces a function

\[ f \in ((X \to U) \times T) \to (X \to V) \]

via the equation

\[ f (h, t) x = a (h x, t). \]

To verify that \( \Phi (\tau, \theta) \) is indeed a natural transformation,

let \( \Phi (\phi, \rho) \in X \to Y, \) and consider the diagram

\[ \begin{array}{ccc}
(X \to \text{Mng } (\tau \to \theta) X) \times \text{Mng } \tau X & \Phi (\tau, \theta) X & \text{Mng } \theta X \\
(\phi \to \text{Mng } (\tau \to \theta) (\phi, \rho)) \times \text{Mng } \tau (\phi, \rho) & \Phi (\tau, \theta) Y \downarrow & \phi \to \text{Mng } \theta (\phi, \rho) \\
(Y \to \text{Mng } (\tau \to \theta) Y) \times \text{Mng } \tau Y & \Phi (\tau, \theta) Y & \text{Mng } \theta Y.
\end{array} \]

Let \( h \in X \to \text{Mng } (\tau \to \theta) X, t \in \text{Mng } \tau X, y \in Y. \) Then

compute:

\[ \Phi (\tau, \theta) Y \circ ((\phi \to \text{Mng } (\tau \to \theta) (\phi, \rho)) \times \text{Mng } \tau (\phi, \rho)) (h, t) y \]

\[ = \Phi (\tau, \theta) Y (\text{Mng } (\tau \to \theta) (\phi, \rho) \circ h \circ \phi, \text{Mng } \tau (\phi, \rho) t) y \]

\[ = \text{Ap } Y (\text{Mng } (\tau \to \theta) (\phi, \rho) (h (\phi y)), \text{Mng } \tau (\phi, \rho) t) \]

\[ = (\text{Ap } Y \circ (\text{Mng } (\tau \to \theta) (\phi, \rho) \times \text{Mng } \tau (\phi, \rho))) (h (\phi y), t). \]
\[ (Mng \circ (\phi,\rho) \circ Ap \ X) \langle h \ (\phi \ y), \ t \rangle, \]

by the naturality of Ap,
\[ = Mng \circ (\phi,\rho) \ (Ap \ X \ (h \ (\phi \ y), \ t)) \]
\[ = Mng \circ (\phi,\rho) \ \ (\phi (\tau,\theta) \ X (h, \ t) \ (\phi \ y)) \]
\[ = ((\phi \Rightarrow Mng \circ (\phi,\rho)) \ \phi (\phi,\theta) X (h, \ t) \ (\phi \ y)) \]
\[ = ((\phi \Rightarrow Mng \circ (\phi,\rho)) \ \phi (\tau,\theta) X (h, \ t)) \ y. \]

This shows that \( \phi (\tau,\theta) \) is natural. Now consider the composition of natural transformations
\[ N (\tau \Rightarrow \theta) \times Mng \tau \xrightarrow{\phi (\tau,\theta)} N \theta \xrightarrow{\Delta \theta} Mng \theta. \]
(This is where \( \Delta \theta \) is used.) By using the abstraction bijection
\[ Ab \varepsilon (N (\tau \Rightarrow \theta) \times Mng \tau) \xrightarrow{\Sigma \Rightarrow P \text{dom}} Mng (\tau \Rightarrow \theta), \]
we define
\[ \Delta (\tau \Rightarrow \theta) \varepsilon N (\tau \Rightarrow \theta) \xrightarrow{\Sigma \Rightarrow P \text{dom}} Mng (\tau \Rightarrow \theta), \]
by
\[ \Delta (\tau \Rightarrow \theta) = Ab (\Delta \theta \circ \phi (\tau,\theta)). \]

This completes the definition of \( \Delta \tau \) for every phrase type \( \tau \).

Do not lose sight of the fact that the purpose of all this is to define
\[ \text{semf ifthenelse}_\tau \]
as the natural transformation which is in canonical correspondence with
\[ \Delta \tau \circ \psi \tau \in \text{B} \tau \xrightarrow{|K|} Mng \tau. \]
Evidence that we have found the correct definition of ifthenelse_{τ} is provided by the following theorem.

**Theorem 7.1:** Let τ, θ ∈ T and b, r, s, m ∈ L.

Let α ∈ A be such that

Type r α = Type s α = τ → θ,

Type b α = bool-exp,

Type m α = τ.

Let

\[ p = [\text{ifthenelse}_{τ} b r s] m \]

and

\[ q = [\text{ifthenelse}_{θ} b [r m] [s m]] \]

Then

Type p α = Type q α = θ, and

Semf p α = Semf q α.

**Proof:** Since this theorem is peripheral to our development, we omit the straightforward, but lengthy, proof. □

We now take up the definition of

\[ \text{semf rec}_{τ} ∈ \text{Env emp}_{T} \rightarrow (\text{Mng } \tau \rightarrow \text{Mng } \tau) \rightarrow \text{Mng } \tau. \]

It suffices to define the corresponding natural transformation

\[ \text{fix}_{τ} ∈ (\text{Mng } \tau \rightarrow \text{Mng } \tau) \rightarrow \text{Mng } \tau. \]

Let X ∈ Ob Σ be a store shape. Define

\[ \text{fix}_{τ} X ∈ (\text{Mng } \tau \rightarrow \text{Mng } \tau) X \rightarrow \text{Mng } \tau X \]

by

\[ \text{fix}_{τ} X \ η = \ Y_{\text{Mng } \tau X} (A_{X} (\eta X) \ 1_{X}^{\Sigma}) \]

where η ∈ hom_{Σ} X × Mng τ \rightarrow Mng τ (recall that
this is the underlying set of \((\text{Mng } \tau \Rightarrow \text{Mng } \tau) \times X\),

\[ Y_{\text{Mng } \tau \times X} \in (\text{Mng } \tau \times X \Rightarrow \text{Mng } \tau \times X) \quad \xrightarrow{P_{\text{dom}}} \quad \text{Mng } \tau \times X \]

is the least fixed point function \((Y_{\text{Mng } \tau \times X} \; h = \bigcup_{n=0}^{\infty} h^n \downarrow_{\text{Mng } \tau \times X})\), and

\(A_{\times} \in \{ (X + X \times \text{Mng } \tau \times X) \quad \xrightarrow{P_{\text{dom}}} \quad \text{Mng } \tau \times X \}

\quad \xrightarrow{\Sigma} \quad (X + X \quad \xrightarrow{\Sigma} \quad \text{Mng } \tau \times X)\).

The domains and codomains for abstraction functions for \(P_{\text{dom}}\) can be partially ordered in an obvious way to make them predomains; it is then easy to show that abstraction functions are continuous. From this observation a routine argument shows that \(\text{fix } \tau\) is a natural transformation. We require a technical lemma.

**Lemma 7.2:** Let \((\phi, \rho) \in X \xrightarrow{\Sigma} Y\) and \(\eta \in \text{hom}_{\Sigma} X \times \text{Mng } \tau \rightarrow \text{Mng } \tau\). For each \(n = 0, 1, 2, \ldots\),

\[ (A_Y ((\text{Mng } \tau \Rightarrow \text{Mng } \tau) \langle \phi, \rho \rangle \eta X) \quad l_{Y}^{n} \downarrow_{\text{Mng } \tau \times Y} \]

\[ = \text{Mng } \tau \langle \phi, \rho \rangle \quad ((A_X \eta X) l_{X}^{n} \downarrow_{\text{Mng } \tau \times X}) \]

**Proof:** We will use induction on \(n\). For \(n = 0\), the assertion becomes

\[ \downarrow_{\text{Mng } \tau \times Y} = \text{Mng } \tau \langle \phi, \rho \rangle \quad \downarrow_{\text{Mng } \tau \times X} \]

which is true because \(\text{Mng } \tau \langle \phi, \rho \rangle\) is strict.

Suppose the lemma is true for \(n = k\). Since \(\eta\) is a natural transformation, we have a commutative diagram
\[(X \to X) \times \text{Mng } \tau X \xrightarrow{\eta X} \text{Mng } \tau X\]

\[\text{hom}_X^{\Sigma} (\phi, \rho) \times \text{Mng } \tau (\phi, \rho) \xrightarrow{} \text{Mng } \tau (\phi, \rho)\]

\[(X \to Y) \times \text{Mng } \tau Y \xrightarrow{\eta Y} \text{Mng } \tau Y\]

Let \(B = \text{Mng } \tau (\phi, \rho) ((A_X \ (\eta X) \ 1_X^\Sigma)^k \ 1_{\text{Mng } \tau X})\).

Then

\[(A_Y \ ((\text{Mng } \tau \to \text{Mng } \tau) \ (\phi, \rho) \ \eta Y) \ 1_Y^\Sigma)^{k+1} \ 1_{\text{Mng } \tau Y}\]

\[= A_Y \ ((\text{Mng } \tau \to \text{Mng } \tau) \ (\phi, \rho) \ \eta Y) \ 1_Y^\Sigma \ B,\]

by the induction hypothesis,

\[= (\text{Mng } \tau \to \text{Mng } \tau) \ (\phi, \rho) \ \eta Y \ (1_Y, B),\]

by the definition of abstraction in \(\text{Pdom}\),

\[= \eta Y ((\phi, \rho), B),\]

by the definition of exponentiation in \(|K|\),

\[= (\eta Y \circ (\text{hom}_X^{\Sigma} (\phi, \rho) \times \text{Mng } \tau (\phi, \rho))) \ (1_X^\Sigma, (A_X \ (\eta X) \ 1_X^\Sigma)^k \ 1_{\text{Mng } \tau X})\]

\[= (\text{Mng } \tau (\phi, \rho) \circ \eta X) \ (1_X^\Sigma, (A_X \ (\eta X) \ 1_X^\Sigma)^k \ 1_{\text{Mng } \tau X}),\]

by the commutativity of the above diagram,

\[= \text{Mng } \tau (\phi, \rho) (\eta X \ (1_X^\Sigma, (A_X \ (\eta X) \ 1_X^\Sigma)^k \ 1_{\text{Mng } \tau X}'))\]

\[= \text{Mng } \tau (\phi, \rho) (A_X \ (\eta X) \ 1_X^\Sigma ((A_X \ (\eta X) \ 1_X^\Sigma)^k \ 1_{\text{Mng } \tau X}'))\]

\[= \text{Mng } \tau (\phi, \rho) ((A_X \ (\eta X) \ 1_X^\Sigma)^{k+1} \ 1_{\text{Mng } \tau X}).\]

\[\square\]

To return to the claim that \(\text{fix}_\tau\) is a natural transformation, suppose \((\phi, \rho) \in X \to Z\), and consider the diagram in \(\text{Pdom}\).
(Mng \tau \Rightarrow Mng \tau) \ X \xrightarrow{\text{fix}_\tau} Mng \tau \times Mng \tau

(Mng \tau \Rightarrow Mng \tau) \ (\phi, \rho) \xrightarrow{\text{fix}_\tau} Mng \tau (\phi, \rho)

(Mng \tau \Rightarrow Mng \tau) \ Z \xrightarrow{\text{fix}_\tau} Mng \tau \times Mng \tau

Let \eta \in \text{hom}_{\Sigma} X \times Mng \tau \xrightarrow{\Sigma = \text{Pdom}} Mng \tau. The computation

(fix_\tau Z \circ (Mng \tau \Rightarrow Mng \tau) (\phi, \rho)) \eta

= \text{fix}_\tau Z ((Mng \tau \Rightarrow Mng \tau) (\phi, \rho) \eta)\eta

= Y_{Mng \tau} Z (A_Z ((Mng \tau \Rightarrow Mng \tau) (\phi, \rho) \eta) \eta Z) \eta Z

= \bigcup_{n=0}^\infty ((A_Z ((Mng \tau \Rightarrow Mng \tau) (\phi, \rho) \eta) \eta Z) \eta Z) \eta Z

= \bigcup_{n=0}^\infty (Mng \tau (\phi, \rho) ((A_X (\eta X) \eta X) \eta Z) \eta Z)

by Lemma 7.2,

= Mng \tau (\phi, \rho) \bigcup_{n=0}^\infty ((A_X (\eta X) \eta X) \eta Z) \eta Z

by the continuity of Mng \tau (\phi, \rho),

= Mng \tau (\phi, \rho) (Y_{Mng \tau} X (A_X (\eta X) \eta X))

= Mng \tau (\phi, \rho) (\text{fix}_\tau X \eta)

= (Mng \tau (\phi, \rho) \circ \text{fix}_\tau X) \eta

shows \text{fix}_\tau is natural.

We must do some preliminary work with the category \Sigma of store shapes before giving the semantics of \text{newvar}_\delta for a data type \delta. We will give some definitions and three lemmas.

The functor

\text{Newshape}_\delta \in \Sigma \rightarrow \Sigma
is defined for \( X \in \text{Ob} \, \Sigma \) by
\[
\text{Newshape}_\delta X = X \times \text{Val} \, \delta
\]
and for \( \langle \phi, \rho \rangle \in X \rightarrow Y \) by
\[
\text{Newshape}_\delta \langle \phi, \rho \rangle = \langle \phi_\delta, \rho_\delta \rangle \in X \times \text{Val} \, \delta \rightarrow Y \times \text{Val} \, \delta,
\]
where
\[
\phi_\delta \in Y \times \text{Val} \, \delta \rightarrow X \times \text{Val} \, \delta
\]
is given by
\[
\phi_\delta \langle y, v \rangle = \langle \phi \, y, v \rangle,
\]
and
\[
\rho_\delta \in X \times \text{Val} \, \delta \rightarrow (Y \times \text{Val} \, \delta \rightarrow Y \times \text{Val} \, \delta)
\]
is given by
\[
\rho_\delta \langle x, v \rangle \langle y, v' \rangle = \langle \rho \, x, y, v \rangle.
\]
It is not too hard to see that \( \langle \phi_\delta, \rho_\delta \rangle \) satisfies (31), (32), and (33) and that \( \text{Newshape}_\delta \) really is a functor.

We will use \( \text{Newshape}_\delta \) to create a local store shape from a global store shape, as must be done upon entering a block containing a declaration of a \( \delta \)-variable.

We posit the existence of a specified element
\[
\text{init} \, \delta \in \text{Val} \, \delta,
\]
which is used as an initial value for a newly declared variable. For each store shape \( X \), this leads to the definition of
\[
\text{enter}_\delta X \in X \rightarrow X \times \text{Val} \, \delta,
\]
given for \( x \in X \) by
\[
\text{enter}_\delta X \, x = \langle x, \text{init} \, \delta \rangle.
\]
We also have, for each store shape $X$,

$$\text{exit}_\delta X \langle x, v \rangle = x.$$  

Obviously, these functions will be used upon entering and exiting from blocks.

**Lemma 7.3:** Let $(\phi, \rho) \in X \to Y$. Then

$$\text{(enter}_\delta Y \to \text{exit}_\delta Y) \circ \rho = \rho \circ \text{exit}_\delta X.$$

Thus, for all $x \in X$, $v \in V$, $y \in Y$,

$$\text{exit}_\delta Y \rho \langle x, v \rangle (\text{enter}_\delta Y y) = \rho (\text{exit}_\delta X \langle x, v \rangle) y.$$

**Proof:** The proof is an easy computation left to the reader. □

The $(\text{Ob } \Sigma)$-indexed family $\text{exit}_\delta$ of functions is intimately related to a natural transformation

$$\text{Exit}_\delta \in \sum_{\Sigma \to \Sigma} \text{Newshape}_\delta,$$

given by

$$\text{Exit}_\delta X = \langle \text{exit}_\delta X, \forall \rho \in X \to X \times \text{Val } \delta \rangle$$

given by

$$\forall \rho X \in X \to (X \times \text{Val } \delta \to X \times \text{Val } \delta)$$

is defined for $x, x' \in X$, $v \in \text{Val } \delta$ by

$$\forall \rho X \langle x, x', v \rangle = \langle x, v \rangle.$$  

Verifications that $\text{Exit}_\delta X$ is a morphism of $\Sigma$ and that $\text{Exit}_\delta$ is natural are left to the reader. The utility of $\text{Exit}_\delta$ is that it contains the information about how to return to the global store shape from the local store shape. There is a way to relate $\text{enter}_\delta$ to a natural transformation, but it involves a slight change in the definition of $\Sigma$, and this
change would entail more complications that it is apparently worth.

**Lemma 7.4:** Let \( (\phi, \rho) \in X \longrightarrow Y \). Then

\[
(\phi_\delta, \rho_\delta) \circ \text{Exit}_\delta X = \text{Exit}_\delta Y \circ (\phi, \rho)
\]

**Proof:** The proof is straightforward. \( \Box \)

It is not surprising that the denotation of newvar involves a "canonical local \( \delta \)-variable for each store shape \( X \)". Recall that

\[
\text{Mng } \delta-\text{var } (X \times \text{Val } \delta) = \text{Mng } \delta-\text{acc } (X \times \text{Val } \delta) \times \text{Mng } \delta-\text{exp}(X \times \text{Val } \delta).
\]

Now for \( X \in \text{Ob } \Sigma \) let

\[
\text{localvar}_\delta X \in \text{Mng } \delta-\text{var } (X \times \text{Val } \delta)
\]

be

\[
\text{localvar}_\delta X = \langle a_\delta, X, e_\delta, X \rangle,
\]

where

\[
a_\delta, X \in \text{Val } \delta \longrightarrow \text{Mng } \text{comm } (X \times \text{Val } \delta) \quad \text{Pdom}
\]

is given for \( v, v' \in V \) and \( x \in X \) by

\[
a_\delta, X(v, v') = (x, v),
\]

and

\[
e_\delta, X : X \times \text{Val } \delta \longrightarrow (\text{Val } \delta)_l \quad \text{Pdom}
\]

is given for \( x \in X, v \in \text{Val } \delta \) by

\[
e_\delta, X(x, v) = v.
\]

Note the use of condition (12) in the proof of the next lemma.

**Lemma 7.5:** Let \( (\phi, \rho) \in X \longrightarrow Y \). Then

\[
\text{Mng } \delta-\text{var } (\phi_\delta, \rho_\delta)(\text{localvar}_\delta X) = \text{localvar}_\delta Y.
\]

**Proof:** We claim

\[
\text{Mng } \delta-\text{acc } (\phi_\delta, \rho_\delta) a_\delta, X = a_\delta, Y.
\]
To see this, let \( v \in \text{Val } \delta \) and \( \langle y, v' \rangle \in Y \in \text{Val } \delta \). Then

\[
\text{Mng } \delta\text{-acc } (\phi_\delta, \rho_\delta) \ a_\delta, x \ v \ (y, v')
\]

\[
= \text{Mng } \text{comm } (\phi_\delta, \rho_\delta) \ (a_\delta, x \ v) \ (y, v')
\]

\[
= \rho_\delta, (a_\delta, x \ v (\phi_\delta (y, v'))) \ (y, v')
\]

\[
= \rho_\delta, (a_\delta, x \ v (\phi_\delta (y, v'))) \ (y, v')
\]

\[
= \rho_\delta, (\phi_\delta (y, v')) \ (y, v')
\]

\[
= (\rho (\phi_\delta) y, v)
\]

\[
= (y, v), \text{ by (12)},
\]

\[
= a_\delta, v \ (y, v'),
\]

which proves the claim.

Furthermore, we claim

\[
\text{Mng } \delta\text{-exp } (\phi_\delta, \rho_\delta) \ e_\delta, x = e_\delta, y.
\]

Let \( \langle y, v \rangle \in Y \times \text{Val } \delta \). We compute:

\[
\text{Mng } \delta\text{-exp } (\phi_\delta, \rho_\delta) \ e_\delta, x \ (y, v)
\]

\[
= e_\delta, x (\phi_\delta (y, v))
\]

\[
= e_\delta, x (\phi_\delta (y, v))
\]

\[
= v
\]

\[
= e_\delta, y \ (y, v).
\]

By combining these two facts we see

\[
\text{Mng } \delta\text{-var } (\phi_\delta, \rho_\delta) \ (\text{localvar}_\delta x)
\]

\[
= (\text{Mng } \delta\text{-acc } (\phi_\delta, \rho_\delta) \times \text{Mng } \delta\text{-exp } (\phi_\delta, \rho_\delta)) \ (a_\delta, x, e_\delta, y)
\]

\[
= (a_\delta, y, e_\delta, y)
\]

\[
= \text{localvar}_\delta y,
\]

as desired. \( \Box \)
We want to define the natural transformation
\[
\text{semf } \text{newvar}_\delta \in \text{Env} \text{emp}_T \quad \longrightarrow \quad (\text{Mng } \delta\text{-var} \Rightarrow \text{Mng comm}) \Rightarrow \text{Mng comm};
\]
so it suffices to define the corresponding natural transformation
\[
nv_\delta \in \text{Mng } \delta\text{-var} \Rightarrow \text{Mng comm} \quad \longrightarrow \quad \text{Mng comm};
\]
Thus, we must give, for \(X \in \text{Ob } \Sigma\),
\[
nv_\delta X \in (\text{Mng } \delta\text{-var} \Rightarrow \text{Mng comm}) \xrightarrow{P\text{dom}} \text{Mng comm } X.
\]
For \(\eta \in \text{hom}_\Sigma X \times \text{Mng } \delta\text{-var} \longrightarrow \text{Mng comm}\) (recall that this is the underlying set of \((\text{Mng } \delta\text{-var} \Rightarrow \text{Mng comm}) X\)), let
\[
nv_\delta X \eta
\]
\[
= (\text{exit}_\delta X)_\bot \circ \eta \left(\text{Newshape}_\delta X\right) \left(\text{Exit}_\delta X, \text{localvar}_\delta X\right) \circ \text{enter}_\delta X,
\]
where
\[
(\text{exit}_\delta X)_\bot \in (X \times \text{Val } \delta)_\bot \xrightarrow{P\text{dom}} X_\bot
\]
is the strict extension of \(\text{exit}_\delta X\). Roughly speaking, we are saying that \(nv_\delta X \eta\) means enter a new block, execute the command derived from \(\eta\) in the context of the new store shape, and finally exit from the new block in a manner that returns the store to its original shape. It is easy to see that \(nv_\delta X\) is continuous.

We must verify that \(nv_\delta\) is a natural transformation. Let \(M\) denote the functor \(\text{Mng } \delta\text{-var} \Rightarrow \text{Mng comm}\). Suppose \((\phi,\rho) \in X \longrightarrow Y\). Consider the commutativity of
Let \( \eta \in \text{hom}^\Sigma X \times \text{Mng}\ \delta\text{-var} \rightarrow \text{Mng comm} \) (the underlying set of \( M X \)), and let \( y \in Y \). It is important to note that

\[
\begin{align*}
\text{hom}^\Sigma X \times (\text{Newshape}_\delta X) \times \\
\text{Mng}\ \delta\text{-var} \ (\text{Newshape}_\delta X) & \to \eta \ (\text{Newshape}_\delta X) \\
\text{hom}^\Sigma \langle \phi_\delta, \rho_\delta \rangle \times \\
\text{Mng}\ \delta\text{-var} \ \langle \phi_\delta, \rho_\delta \rangle & \to \eta \ (\text{Newshape}_\delta Y) \\
\text{hom}^\Sigma X \times (\text{Newshape}_\delta Y) \times \\
\text{Mng}\ \delta\text{-var} \ (\text{Newshape}_\delta Y) & \to \eta \ (\text{Newshape}_\delta Y)
\end{align*}
\]

commutes because \( \eta \) is a natural transformation. Then

\[
(nv_\delta Y \circ (1 \times M \langle \phi, \rho \rangle)) \eta \ y
= nv_\delta Y (M \langle \phi, \rho \rangle \eta) \ y
= ((\text{exit}_\delta Y)_\perp
\circ M \langle \phi, \rho \rangle \eta \ (\text{Newshape}_\delta Y) \ (\text{Exit}_\delta Y, \text{localvar}_\delta Y)
\circ \text{enter}_\delta Y) \ y
= ((\text{exit}_\delta Y)_\perp
\circ \eta \ (\text{Newshape}_\delta Y) \ (\text{Exit}_\delta Y \circ \langle \phi, \rho \rangle), \text{localvar}_\delta Y)
\circ \text{enter}_\delta Y) \ y,
\]
by the definition of \( \Rightarrow \) for \( |K| \),

\[
\begin{align*}
&= (\text{exit}_\delta Y) \perp \\
&\circ \eta (\text{Newshape}_\delta Y) (\langle \phi_\delta, \rho_\delta \rangle \circ \text{Exit}_\delta X, \text{localvar}_\delta Y) \\
&\circ \text{enter}_\delta Y) Y,
\end{align*}
\]

by Lemma 7.4,

\[
\begin{align*}
&= (\text{exit}_\delta Y) \perp \\
&\circ \eta (\text{Newshape}_\delta Y) \\
&\quad (\langle \text{hom}^\Sigma X (\phi_\delta, \rho_\delta) \times \text{Mng} \delta-\text{var} (\phi_\delta, \rho_\delta) \rangle (\text{Exit}_\delta X, \text{localvar}_\delta X)) \\
&\circ \text{enter}_\delta Y) Y,
\end{align*}
\]

by Lemma 7.5

\[
\begin{align*}
&= (\text{exit} Y) \perp \\
&\circ \text{Mng} \, \text{comm} (\phi_\delta, \rho_\delta) (\eta (\text{Newshape}_\delta X) (\text{Exit}_\delta X, \text{localvar}_\delta X) \\
&\circ \text{enter}_\delta Y) Y
\end{align*}
\]

\[
\begin{align*}
&= (\text{exit}_\delta Y) \perp \\
&\quad (\rho_\delta, \perp (\eta (\text{Newshape}_\delta X) (\text{Exit}_\delta X, \text{localvar}_\delta X) (\phi_\delta (\text{enter}_\delta Y) y)))(\text{enter}_\delta Y) Y)
\end{align*}
\]

\[
\begin{align*}
&= \rho \perp (\text{exit}_\delta X) \perp \\
&\quad (\eta (\text{Newshape}_\delta X) (\text{Exit}_\delta X, \text{localvar}_\delta X) \\
&\quad (\phi_\delta (\text{enter}_\delta Y) y)))(\text{enter}_\delta X (\phi Y)) Y,
\end{align*}
\]

by Lemma 7.3,
\[ = (\text{Mng comm} (\phi, \delta) \circ \nu_\delta X) \eta_Y, \]

and this proves \( \nu_\delta \) is a natural transformation.
CHAPTER VIII
DESUGARED ALGOL: CONTINUATION SEMANTICS

In this chapter we expand the language given in the preceding chapter to include goto's and labels. The main novelties consist of the introduction of the primitive phrase type \texttt{compl} (for "completion"), whose set of meanings in the context of a store shape are what are normally called continuations, and of the removal of \texttt{comm} from the collection of primitive phrase types. Our point of view is now that \texttt{comm} is just an abbreviation for the procedural phrase type \texttt{compl} $\Rightarrow$ \texttt{compl}.

As in Chapter VII we assume we are given

1. $\mathcal{D}$, a poset of data types,
2. $\mathcal{O}$, a collection of data-type operators,
3. $\text{arity} \in \mathcal{O} \rightarrow (\text{Ob } \mathcal{D})^*$ and $\text{res} \in \mathcal{O} \rightarrow \text{Ob } \mathcal{D}$,
4. $\text{Val} \in \mathcal{D} \rightarrow \text{Set}$, a functor,
5. $\text{m}_\mathcal{O}$, an appropriate interpretation for the operators in $\mathcal{O}$, and
6. $\text{Id}$, an infinite set of identifiers.

We first diverge from Chapter VII by changing $\mathcal{P}$, the poset of primitive phrase types. At first, the change appears superficial: \texttt{comm} is just replaced by \texttt{compl}. Thus
\[ \text{Ob } P = \{ \text{compl} \} \]
\[ \cup \{ \delta \text{-exp} \mid \delta \in \mathcal{D} \} \]
\[ \cup \{ \delta \text{-acc} \mid \delta \in \mathcal{D} \} \]
\[ \cup \{ \delta \text{-var} \mid \delta \in \mathcal{D} \}, \]

and the partial order on \( P \) is described in exactly the same way as was the partial order on the poset of primitive phrase types in Chapter VII.

As before, \( P \) generates \( T \), a free type algebra, and \( \text{Ob } T \) is the disjoint union

\[ \{ \text{ns} \} \cup P \cup \{ \tau \to \theta \mid \tau, \theta \in T \}. \]

Throughout this chapter we consider \text{comm} to be an abbreviation for \text{compl} \to \text{compl}. So all the phrase types from Chapter VII, and more, are available to us. In a sense, we have added the "square root" of \text{comm} to the type algebra; we leave it to some algebraist to formalize that intuition.

Again, the language \( L \) is a free \( \Lambda \)-algebra generated by a typed set \( G \) of linguistic constants with typing function

\[ \text{type} \in G \to \text{Ob } T. \]

As before, \( G \) is the disjoint union of \( \text{Op} \) and \( \text{Op}' \). \( \text{Op} \) and the restriction of type to \( \text{Op} \) are the same as in Chapter VII. However, we add two new linguistic constants, \( \text{halt} \) and \( \text{goto} \), to \( \text{Op}' \), so that \( \text{Op}' \) is given by the following table, where \( \delta \in \mathcal{D} \) and \( \tau \in T \).
\( g \in O^p \quad \text{type } g \)

- skip \( \quad \text{comm} \)
- ; \( \quad \text{comm} \Rightarrow \text{comm} \Rightarrow \text{comm} \)
- \(:=\ \delta \quad \delta\text{-acc} \Rightarrow \delta\text{-exp} \Rightarrow \text{comm} \)
- ifthenelse \( \tau \quad \text{bool-exp} \Rightarrow \tau \Rightarrow \tau \Rightarrow \tau \)
- rec \( \tau \quad (\tau \Rightarrow \tau) \Rightarrow \tau \)
- newvar \( \delta \quad (\delta\text{-var} \Rightarrow \text{comm}) \Rightarrow \text{comm} \)
- halt \( \quad \text{compl} \)
- goto \( \quad \text{compl} \Rightarrow \text{comm} \)

Intuitively, halt is the completion to which well-formed programs, which are commands, are applied when they are executed. The constant goto takes a completion ("label value") and turns it into a command; the resulting command when applied to another completion ignores it and passes control to the completion to which the goto was applied.

To see how to desugar blocks with label definitions, some discussion is necessary. Regard

let \( x:\tau \) be \( m \) in \( n \)

as syntactic sugar for

\([ (\lambda x:\tau.n) m ] , \)

and regard

letrec \( x:\tau \) be \( m \) in \( n \)

as syntactic sugar for

\([ (\lambda x:\tau.n) (\text{rec}\_\tau (\lambda x:\tau.m)) ] . \)
Multiple letrec's can now be desugared (see Reynolds [6]).

For instance, regard

\[
\text{letrec } x_1: \tau_1 \text{ be } m_1 \& \\
    \quad x_2: \tau_2 \text{ be } m_2 \\
\text{in } n
\]

as

\[
\text{letrec } x_1: \tau_1 \text{ be (letrec } x_2: \tau_2 \text{ be } m_2 \text{ in } m_1) \\
\text{in (letrec } x_2: \tau_2 \text{ be } m_2 \text{ in } n).
\]

The block with label definitions

\[
(c_0 ; \ell_1: c_1 ; \ell_2: c_2 ; \ldots ; \ell_n: c_n),
\]

which has phrase type \text{comm} = \text{compl} \rightarrow \text{compl} may be regarded as

\[
\lambda e: \text{compl. letrec } \ell_1: \text{compl} \text{ be } \llbracket c_1 \ell_2 \rrbracket \& \\
            \ell_2: \text{compl} \text{ be } \llbracket c_2 \ell_3 \rrbracket \& \\
            \quad \ldots \\
            \ell_n: \text{compl} \text{ be } \llbracket c_n e \rrbracket \\
\text{in } \llbracket c_0 \ell_1 \rrbracket.
\]

(Intuitively, \( e \) denotes the "normal" exit from the block.
The labels \( \ell_1, \ldots, \ell_n \) are "used" if commands of the form
goto \( \ell_1 \) are executed during execution of the block.)

Hence, a block such as

\[
(c ; \ell: \text{skip})
\]

that contains a single label, which is used via the goto \( \ell \)
command to accomplish an "abnormal" exit from the block, may
be desugared by means of the following sequence of steps:
(c ; ℓ:skip),
λe:compl. (letrec ℓ:compl be [skip e] in [c ℓ]),
λe:compl. (letrec ℓ:compl be e in [c ℓ]),
λe:compl. (let ℓ:compl be e in [c ℓ]),
λe:compl. ((λℓ:compl. [c ℓ] e),
λℓ:compl. [c ℓ].

The method by which one proceeds from one step to the next above is rather informal. (This indicates that more work needs to be done on the nature of desugaring.) As long as we are being informal, when c contains no occurrences of ℓ, we can add the step

c

to the above list.

The semantics of this language ℓ involves specifying

(1) a type algebra K derived from a Cartesian closed category,

(2) a functor \( \text{mng} \in P \rightarrow |K| \), which serves to give meaning objects for each primitive phrase type, and which extends to a uniquely defined type algebra homomorphism \( \text{mng} \in T \overset{\text{TypeAlg}}{\rightarrow} K \),

(3) for each \( g \in G \), an arrow of \(|K|\)

\[ \text{semf} \ g \in \text{Env} \, \text{emp}_{T} \overset{\text{TypeAlg}}{\rightarrow} \text{Mng} \ (\text{type} \ g). \]

In this chapter we will use the same type algebra K as was used in Chapter VII. Thus, K is the type algebra canonically associated with the Cartesian closed category
$|K|$, the full subcategory of $\Sigma \Rightarrow P_{dom}$ whose collection of objects consists of those functors which give domains when applied to store shapes and which give strict, continuous functions when applied to expansions.

Turning now to the definition of $\text{mng} \in \Omega \Rightarrow |K|$, we assume that we are given $\emptyset$, a domain of "outputs". The minimal element of $\emptyset$ is to be viewed as the undefined output. Perhaps $\emptyset$ might be the flat poset consisting of $\bot$ and all strings of data type values. Then again, perhaps not. We do not need to be more explicit about the story of $\emptyset$.

We must give functors

$m\text{ng}{}^\text{compl}, \text{mng}{}^\delta\text{-exp}, \text{mng}{}^\delta\text{-acc}, \text{mng}{}^\delta\text{-var} \in \Sigma \Rightarrow P_{dom}$,

where $\delta \in \Omega$, and show they are objects of $|K|$

Let

$m\text{ng}{}^\text{compl} = (U -) \Rightarrow \emptyset$,

where $U \in \Sigma \Rightarrow P_{dom}^{op}$ is the forgetful functor which forgets the second component of an expansion. Thus, for a store shape $X \in \text{Ob } \Sigma$,

$m\text{ng}{}^\text{compl} X = X \Rightarrow \emptyset$,

the set of continuations (functions from stores to outputs) appropriate to the store shape $X$. Since $\emptyset$ is a domain, $X \Rightarrow \emptyset$ is a domain, and $m\text{ng}{}^\text{compl}$ applied to store shapes yields domains.

Suppose $\langle \phi, \rho \rangle \in X \Rightarrow Y$. Then

$m\text{ng}{}^\text{compl} \langle \phi, \rho \rangle = \phi = l_\emptyset \in X \Rightarrow \emptyset \Rightarrow \emptyset$. 
Let $y \in Y$. The computation

$$\begin{align*}
(\phi \Rightarrow 1_0) \downarrow_{x \Rightarrow 0} y \\
= (1_0 \circ \downarrow_{x \Rightarrow 0} \circ \phi) \ y \\
= \downarrow_{x \Rightarrow 0} (\phi \ y) \\
= 1_0 \\
= \downarrow_{y \Rightarrow 0} \ y
\end{align*}$$

shows that $\text{mng compl} (\phi, \rho)$ is strict. Hence $\text{mng compl}$ is an object of $|K|$. 

Suppose $\delta \in \mathcal{D}$; then $\text{mng } \delta\text{-exp} \in \Sigma \rightarrow P\text{dom}$ is defined just as in Chapter VII, i.e.

$$\text{mng } \delta\text{-exp} = (U \rightarrow) \Rightarrow (\text{Val } \delta) \downarrow_{1},$$

so that for $X \in \text{Ob } \Sigma$,

$$\text{mng } \delta\text{-exp } X = X \Rightarrow (\text{Val } \delta) \downarrow_{1}$$

(This leads to the idea that a completion is an "output expression," but we will say no more about this intuition.)

Next we will define $\text{mng } \delta\text{-acc}$ for $\delta \in \mathcal{D}$. Since we can't talk about $\text{Mng compl}$ until the definition of $\text{mng}$ is complete, we introduce

$$\text{Cmd } = \text{mng compl } \Rightarrow \text{mng compl},$$

where the heavy arrow is the internal hom functor in $|K|$. Mimicking Chapter VII, let

$$\text{mng } \delta\text{-acc} \in \Sigma \rightarrow P\text{dom}$$

be

$$\text{mng } \delta\text{-acc } = \text{Val } \delta \Rightarrow (\text{Cmd } \rightarrow),$$
where this heavy arrow is exponentiation for $P_{dom}$. Hence, for $X \in \text{Ob} \Sigma$,

\[ \text{mng } \delta \text{-acc } X = \text{Val } \delta = \text{Cmd } X. \]

Just as in Chapter VII, mng $\delta$-acc is an object of $|K|$.

For $\delta \in \mathcal{D}$, the definition of mng $\delta$-var follows the pattern laid down in the last chapter. Thus

\[ \text{mng } \delta \text{-var } = \text{mng } \delta \text{-acc } \times \text{mng } \delta \text{-exp }, \]

where $\times$ is the binary product in $|K|$. Thus, for a store shape $X$,

\[ \text{mng } \delta \text{-var } X = \text{mng } \delta \text{-acc } X \times \text{mng } \delta \text{-exp } X. \]

The definition of mng on primitive phrase types (objects of $P$) is now complete. To define mng on implicit conversions between primitive phrase types (arrows of $P$), we may use the definition for the corresponding entity in Chapter VII, taking care to make the trivial change of using Cmd rather than mng $\text{comm}$. This fully defines mng $\in P \rightarrow |K|$ and also defines $\text{Mng } \in T \xrightarrow{\text{TypeAlg}} K$, the type algebra homomorphism whose restriction to $P$ is mng.

The final item on the agenda is giving the semantics of the generating set $G$ of linguistic constants.

First of all, semf $g$ where $g \in G$ is a data type operator (i.e. $g \in \text{Op}$) can be defined the same way as in Chapter VII. The discussion there carries over verbatim.
Next, consider skip. We must give

\[ \text{semf \ skip} \in \text{Env \ emp} \Rightarrow |K| \Rightarrow \text{Mng \ comm} \]

where \( \text{Mng \ comm} = \text{Mng \ compl} \Rightarrow \text{Mng \ compl} \). By using the Cartesian closed category structure of \( |K| \), it suffices to give the corresponding natural transformation

\[ \text{skp} \in \text{Mng \ compl} \Rightarrow |K| \Rightarrow \text{Mng \ compl}. \]

The obvious definition is

\[ \text{skp} = \text{l}_{\text{Mng \ compl}}. \]

On the list of generators we next find \( ; \). We wish to give

\[ \text{semf} : \in \text{Env \ emp} \Rightarrow |K| \Rightarrow \text{Mng \ comm} \Rightarrow \text{Mng \ comm} \Rightarrow \text{Mng \ comm}, \]

which we do by giving the corresponding natural transformation

\[ \text{sc} \in \text{Mng \ comm} \times \text{Mng \ comm} \times \text{Mng \ compl} \Rightarrow |K| \Rightarrow \text{Mng \ compl}. \]

(Recall that \( \text{Mng \ comm} = \text{Mng \ compl} \Rightarrow \text{Mng \ compl} \).) Let

\[ \text{Ap} \in \text{Mng \ comm} \times \text{Mng \ compl} \Rightarrow |K| \Rightarrow \text{Mng \ compl} \]

denote the application arrow. Take \( \text{sc} \) to be such that

\[
\begin{array}{ccc}
\text{Mng \ comm} \times \text{Mng \ comm} \times \text{Mng \ compl} & \xrightarrow{\text{sc}} & \text{Mng \ compl} \\
\downarrow \text{l} \times \text{Ap} & & \downarrow \text{Ap} \\
\text{Mng \ comm} \times \text{Mng \ compl} & & \text{Mng \ compl}
\end{array}
\]
commutes. There is a deep difference here between direct and continuation semantics. Here the second command is applied first. Curiously, regarding \texttt{comm} as a procedural phrase type has made it possible to give the semantics of \_; in a way that involves many fewer verification arguments than in Chapter VII. We are making strong and efficient use of the connection between \_; and application in a Cartesian closed category.

Suppose \( \delta \in \mathcal{C} \). We will define

\[
\text{semf} := \delta \in \text{Env} \left( \left| \delta \right| \right) \rightarrow \text{Mng \_acc} \rightarrow \text{Mng \_exp} \rightarrow \text{Mng \ comm}
\]

by defining the corresponding natural transformation

\[
\text{assign}_\delta \in \text{Mng \_acc} \rightarrow \text{Mng \_exp} \rightarrow \text{Mng \ compl} \quad \left| \delta \right| \rightarrow \text{Mng \ compl}.
\]

Let \( X \in \text{Ob} \mathcal{C} \) be a store shape. We seek

\[
\text{assign}_\delta \, X \in (\text{Val} \, \delta \rightarrow \text{Mng \ compl} \, X) \times (X \rightarrow (\text{Val} \, \delta) \rightarrow) \times (X \rightarrow (\text{Val} \, \delta) \rightarrow) \rightarrow (X \rightarrow (\text{Val} \, \delta) \rightarrow)
\]

Suppose \( a \in \text{Val} \, \delta \rightarrow \text{Mng \ compl} \, X, \, e \in X \rightarrow \text{Val} \, \delta \rightarrow \rightarrow \rightarrow \rightarrow \), and \( k \in X \rightarrow 0 \). Let \( a_\perp \) denote the strict extension of \( a \).

For each \( x \in X \), note that

\[
a_\perp \, (e \, x) \in \text{hom} \left( X \times \text{Mng \ compl} \left| \delta \right| \rightarrow \text{Mng \ compl} \right).
\]

Therefore, we may define

\[
\text{assign}_\delta \, (a, \, e, \, k) \in X \rightarrow \text{Mng \ compl} \left| \delta \right| \rightarrow \text{Mng \ compl} \rightarrow 0
\]

by

\[
\text{assign}_\delta \, (a, \, e, \, k) \, (x) = a_\perp \, (e \, x) \, (l_X \, k) \, x, \text{ where } x \in X.
\]

Note the two occurrences of \( x \) on the right-hand side of the
equation, indicating that the store used to produce a value is the store to be altered by the assignment command. The proof that assign_δ X is continuous is straightforward. We must prove that assign_δ is a natural transformation, so suppose (φ, ρ) ∈ X → Y, and consider the diagram:

\[
\begin{array}{c}
(\text{Val}_δ \rightarrow \text{Mng comm } X) \times (X \rightarrow \text{Val}_δ)_ι \times (X \Rightarrow 0) \xrightarrow{\text{assign}_δ X} (X \Rightarrow 0) \\
(\text{lMng comm} (φ, ρ)) \times (φ \Rightarrow 1) \times (φ \Rightarrow 1) \xrightarrow{\phi \Rightarrow 1} \\
(\text{Val}_δ \rightarrow \text{Mng comm } Y) \times (Y \rightarrow \text{Val}_δ)_ι \times (Y \Rightarrow 0) \xrightarrow{\text{assign}_δ Y} (Y \Rightarrow 0).
\end{array}
\]

For y ∈ Y, compute:

\[
\begin{align*}
(\text{assign}_δ Y \circ ((\text{lMng comm} (φ, ρ)) \times (φ \Rightarrow 1) \times (φ \Rightarrow 1))) (a, e, k) & \cdot y \\
= \text{assign}_δ Y \langle \text{Mng comm} (φ, ρ) \circ a, e \circ φ, k \circ φ \rangle \cdot y \\
= (\text{Mng comm} (φ, ρ) \circ a) \_ι ((e \circ φ) \cdot y) \langle Y \_ι Y, k \circ φ \rangle \cdot y \\
= (\text{Mng comm} (φ, ρ) \circ a) \_ι ((e \circ φ) \cdot y) \langle Y \_ι Y, k \circ φ \rangle \cdot y,
\end{align*}
\]

because Mng comm (φ, ρ) is strict,

\[
= \text{Mng comm} (φ, ρ) \circ a) \_ι (e (φy)) \langle Y \_ι Y, k \circ φ \rangle \cdot y
= a) \_ι (e (φy)) \langle (φ, ρ), k \circ φ \rangle \cdot y,
\]

since Mng comm = Mng compl → Mng compl,

\[
= a) \_ι (e (φy)) X \langle Y \_ι Y, k \rangle (φy),
\]

by the naturality of a) \_ι (e (φ y))

\[
= \text{assign}_δ X (a, e, k) (φ y)
= (φ \Rightarrow 1) (\text{assign}_δ X (a, e, k)) y
= (((φ \Rightarrow 1) \circ \text{assign}_δ X) (a, e, k)) y,
\]

and this proves assign_δ is natural.
In this chapter we take a different approach to the semantics of ifthenelse, where \( \tau \in \mathcal{T} \). As before, we seek to define for each \( \tau \in \mathcal{T} \)

\[
\text{semf ifthenelse}_\tau \in \text{Env} \quad \text{emp}_\tau \quad \text{Mng bool-exp} \quad \text{Mng} \quad \tau \quad \text{Mng} \quad \tau
\]

and we do this by defining the corresponding

\[
\text{cond}_\tau \in \text{Mng} \quad \text{bool-exp} \times \text{Mng} \quad \tau \times \text{Mng} \quad \tau \quad \text{Mng} \quad \tau
\]

However, at this point the treatment diverges from that in the last chapter.

We first take up how to define \( \text{cond}_\tau \) when \( \text{cond}_\emptyset \) is known. Let

\[
\pi_{\text{forget1}}, \pi_{\text{forget2}} \in \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng} \quad \tau \quad \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng} \quad \tau
\]

denote the projections obtained by forgetting the appropriate \( \text{Mng}(\tau \Rightarrow \emptyset) \) factor in the domain.

Let

\[
\text{Ap} \in \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng} \quad \tau \quad \text{Mng} \quad \emptyset
\]

be the application arrow in \( |K| \). Then

\[
\text{Ap} \circ \pi_{\text{forget1}}, \text{Ap} \circ \pi_{\text{forget2}} \in \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng} \quad \tau \quad \text{Mng} \quad \emptyset.
\]

From the universal mapping property for products we obtain

\[
\langle \text{Ap} \circ \pi_{\text{forget1}}, \text{Ap} \circ \pi_{\text{forget2}} \rangle \in \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng}(\tau \Rightarrow \emptyset) \times \text{Mng} \quad \tau \quad \text{Mng} \quad \emptyset \times \text{Mng} \quad \emptyset.
\]

Now define \( \text{cond}_\tau \) so that the diagram
\[
\begin{align*}
\text{Mng bool-exp} & \times \text{Mng (\tau \Rightarrow \theta)} \times \text{Mng (\tau \Rightarrow \theta)} \times \text{Mng} \tau \\
\text{cond}_{\tau \Rightarrow \theta} & \\
1 \times (\text{Ap} \circ \pi_{\text{forget}1}, \text{Ap} \circ \pi_{\text{forget}2}) & \\
\text{Mng} \theta & \\
\text{cond}_{\theta} & \\
\text{commutes in } |K|.
\end{align*}
\]
Thus assuming that \( \text{cond}_{\theta} \) is known, we can define
\[
\text{cond}_{\tau \Rightarrow \theta} \in \text{Mng bool-exp} \times \text{Mng (\tau \Rightarrow \theta)} \times \text{Mng (\tau \Rightarrow \theta)} \xrightarrow{|K|} \text{Mng} \tau \Rightarrow \text{Mng} \theta
\]
to be the abstraction of
\[
\text{cond}'_{\tau \Rightarrow \theta} \in \text{Mng bool-exp} \times \text{Mng (\tau \Rightarrow \theta)} \times \text{Mng (\tau \Rightarrow \theta)} \times \text{Mng} \tau \xrightarrow{|K|} \text{Mng} \theta.
\]
So to define \( \text{cond}_{\tau} \) for all phrase types \( \tau \), it suffices to define \( \text{cond}_{\pi} \) for primitive phrase types \( \tau \). We now proceed to do this.

(1) \( \pi = \text{compl} \). Let \( X \in \text{Ob} \Sigma \) be a store shape. Define
\[
\text{cond}_{\text{compl} \ X} \in \text{Mng bool-exp} \times \text{Mng comp} X \times \text{Mng comp} X \times \text{Mng compl} X \xrightarrow{\text{Pdom}} \text{Mng compl} X
\]
by
\[
\text{cond}_{\text{compl} \ X} (b, k_1, k_2) = \begin{cases} 
k_1 x & \text{if } b x = \text{true} \\
k_2 x & \text{if } b x = \text{false} \\
1 & \text{if } b x = 1,
\end{cases}
\]
where \( b \in X \xrightarrow{\text{Pdom}} \text{Val bool} \), \( k_1, k_2 \in X \xrightarrow{\text{Pdom}} 0 \), \( x \in X \).

The proof that \( \text{cond}_{\text{compl} \ X} \) is continuous is routine.
We must check that \( \text{cond}_{\text{compl}} \) is a natural transformation, so let \( (\phi, \rho) \in X \xrightarrow{\xi} Y \) be an expansion and consider the diagram

\[
\begin{array}{ccc}
(X \Rightarrow (\text{Val bool} ) \bot ) \times (X \Rightarrow 0) \times (X \Rightarrow 0) & \xrightarrow{\text{cond}_{\text{compl}} X} & (X \Rightarrow 0) \\
(\phi \Rightarrow 1) \times (\phi \Rightarrow 1) \times (\phi \Rightarrow 1) & \xrightarrow{\text{cond}_{\text{compl}} Y} & (Y \Rightarrow 0) \\
(Y \Rightarrow (\text{Val bool} ) \bot ) \times (Y \Rightarrow 0) \times (Y \Rightarrow 0) & \xrightarrow{\text{cond}_{\text{compl}} Y} & (Y \Rightarrow 0) \\
\end{array}
\]

For \( y \in Y \), compute:

\[
(\text{cond}_{\text{compl}} Y \circ (\phi \Rightarrow 1) \times (\phi \Rightarrow 1) \times (\phi \Rightarrow 1)) \langle b, k_1, k_2 \rangle y
\]

\[
= \text{cond}_{\text{compl}} Y \langle b \circ \phi, k_1 \circ \phi, k_2 \circ \phi \rangle y
\]

\[
= \begin{cases} 
  k_1 (\phi y) & \text{if } b(\phi y) = \text{true} \\
  k_2 (\phi y) & \text{if } b(\phi y) = \text{false} \\
  1 & \text{if } b(\phi y) = 1 
\end{cases}
\]

\[
= \text{cond}_{\text{compl}} X \langle b, k_1, k_2 \rangle (\phi y)
\]

\[
= (\text{cond}_{\text{compl}} X \langle b, k_1, k_2 \rangle \circ \phi) y
\]

\[
= ((\phi \Rightarrow 1) \circ \text{cond}_{\text{compl}} X) \langle b, k_1, k_2 \rangle y
\]

so the diagram commutes, and \( \text{cond}_{\text{compl}} \) is indeed natural.

(2) \( \pi = \delta\text{-exp} \) where \( \delta \in \mathcal{D} \). For \( X \in \text{Ob } \Sigma \), define

\[
\text{cond}_{\delta\text{-exp}} X \in
\]

\[
\text{Mng } \text{bool-exp} X \times \text{Mng } \delta\text{-exp} X \times \text{Mng } \delta\text{-exp} X \xrightarrow{\text{Pdom}} \text{Mng } \delta\text{-exp} X
\]

by
\[ \text{cond}_{\delta-\exp} \ x \ (b, \ e_1, \ e_2) x = \begin{cases} \ e_1 x & \text{if } b x = \text{true} \\ \ e_2 x & \text{if } b x = \text{false} \\ 1 & \text{if } b x = 1 \end{cases} \]

where \( b \in X \xrightarrow{Pdom} (\text{Val bool})_1 \), \( e_1, \ e_2 \in X \xrightarrow{Pdom} (\text{Val} \delta)_1 \), \( x \in X \). The necessary verifications for \( \text{cond}_{\delta-\exp} \) are completely parallel to those for \( \text{cond}_{\text{compl}} \).

(3) \( \pi = \delta-\text{acc} \) where \( \delta \in D \). Note that the remarks made earlier make it clear that \( \text{cond}_{\text{comm}} = \text{cond}_{\text{compl} \Rightarrow \text{compl}} \) is defined. Suppose \( X \in \text{Ob} \: \Sigma \) is a store shape. We can define

\[ \text{cond}_{\delta-\text{acc}} X \epsilon \]

\[ \text{Mng} \: \text{bool-exp} \ x \times \text{Mng} \: \delta-\text{acc} X \times \text{Mng} \: \delta-\text{acc} X \xrightarrow{Pdom} \text{Mng} \: \delta-\text{acc} X \]

by

\[ \text{cond}_{\delta-\text{acc}} X \ (b, \ a_1, \ a_2) v = \text{cond}_{\text{comm}} X \ (b, \ a_1 v, \ a_2 v) \]

where \( b \in X \xrightarrow{Pdom} (\text{Val bool})_1 \), \( a_1, \ a_2 \in \text{Val} \delta \xrightarrow{Pdom} \text{Mng} \: \text{comm} X \) and \( v \in \text{Val} \delta \). It is easy to see that \( \text{cond}_{\delta-\text{acc}} X \) is continuous. To see that \( \text{cond}_{\delta-\text{acc}} \) is a natural transformation, let \( (\phi, \rho) \in \Sigma \xrightarrow{\times} \text{Y} \) and consider the commutativity of the diagram.
\[(X \rightarrow (\text{Val bool} \uparrow)) \times (\text{Val } \delta \rightarrow \text{Mng } \text{comm } X) \times (\text{Val } \delta \rightarrow \text{Mng } \text{comm } X)\]

\[\text{cond}_{\delta \text{-acc}} X\]

\[\text{Val } \delta \rightarrow \text{Mng } \text{comm } X\]

\[(Y \rightarrow (\text{Val bool} \uparrow)) \times (\text{Val } \delta \rightarrow \text{Mng } \text{comm } Y) \times (\text{Val } \delta \rightarrow \text{Mng } \text{comm } Y)\]

\[\text{cond}_{\delta \text{-acc}} Y\]

\[\text{Val } \delta \rightarrow \text{Mng } \text{comm } Y\]

To establish this, we compute
(\text{cond}_{\delta}\text{-acc} \ (Y \circ ((\phi \Rightarrow 1) \times (1 \Rightarrow \text{Mng comm} (\phi, \rho))) \times

(1 \Rightarrow \text{Mng comm} \phi, \rho)) \ b, a_1, a_2) \ v

= \text{cond}_{\delta}\text{-acc} \ (Y \ b\phi, \text{Mng comm} (\phi, \rho) \ a_1, \text{Mng comm} (\phi, \rho) \ a_2) \ v

= \text{cond}_{\text{comm}} \ Y \ b\phi, \text{Mng comm} (\phi, \rho) (a_1 \ v), \text{Mng comm} (\phi, \rho) (a_2 \ v)

= \text{Mng comm} (\phi, \rho) \ (\text{cond}_{\text{comm}} X (b, a_1 \ v, a_2 \ v))

by the naturality of \text{cond}_{\text{comm}}

= \text{Mng comm} (\phi, \rho) \ (\text{cond}_{\delta}\text{-acc} X (b, a_1, a_2) \ v)

= (\text{Mng comm} (\phi, \rho) \circ \text{cond}_{\delta}\text{-acc} X (b, a_1, a_2) \ v

= (1 \Rightarrow \text{Mng comm} (\phi, \rho)) \ (\text{cond}_{\delta}\text{-acc} X (b, a_1, a_2)) \ v

= (1 \Rightarrow \text{Mng comm} (\phi, \rho)) \circ \text{cond}_{\delta}\text{-acc} X (b, a_1, a_2) \ v.

Hence \text{cond}_{\delta}\text{-acc} is a natural transformation.

(4) \ \pi = \delta\text{-var} \ where \ \delta \in \mathcal{D}. \ Recall \ that

\text{Mng} \ \delta\text{-var} = \text{Mng} \ \delta\text{-acc} \times \text{Mng} \ \delta\text{-exp}.

The projection onto the first (respectively second) component of \text{Mng} \ \delta\text{-var} induces a projection

\pi_{\delta\text{-acc}} \in \text{Mng bool-exp} \times \text{Mng} \ \delta\text{-var} \times \text{Mng} \ \delta\text{-var}

\overset{K}{\longrightarrow} \text{Mng} \ \text{bool-exp} \times \text{Mng} \ \delta\text{-acc} \times \text{Mng} \ \delta\text{-acc}

(respectively
\[ \pi_{\delta-\text{exp}} \in \text{Mng bool-exp} \times \text{Mng } \delta-\text{var} \times \text{Mng } \delta-\text{var} \]

\[ \Downarrow_{K}^{\text{Mng bool-exp} \times \text{Mng } \delta-\text{exp} \times \text{Mng } \delta-\text{exp}} \]

We may compose the projections \( \pi_{\delta-\text{acc}} \) and \( \pi_{\delta-\text{exp}} \) with \( \text{cond}_{\delta-\text{acc}} \) and \( \text{cond}_{\delta-\text{exp}} \). Let the natural transformation

\[ \text{cond}_{\delta-\text{var}} \in \text{Mng bool-exp} \times \text{Mng } \delta-\text{var} \times \text{Mng } \delta-\text{var} \]

\[ \Downarrow_{K}^{\text{Mng } \delta-\text{acc} \times \text{Mng } \delta-\text{exp}} \]

be

\[ \text{cond}_{\delta-\text{var}} = (\text{cond}_{\delta-\text{acc}} \circ \pi_{\delta-\text{acc}}, \text{cond}_{\delta-\text{exp}} \circ \pi_{\delta-\text{exp}}) \]

the arrow satisfying the obvious universal mapping property. This finishes the definition of \( \text{semf ifthenelse}_\tau \), where \( \tau \in T \).

The semantics of \( \text{rec}_\tau \) is the same for both continuation semantics and direct semantics. The definition of \( \text{semf rec}_\tau \), where \( \tau \in T \), given in Chapter VII can be used here verbatim.

In Chapter VII we defined some objects related to the category of \( \Sigma \) of store shapes. Some of these entities, which were used in giving direct semantics for newvar\( _\delta \), are again useful for giving continuation semantics for newvar\( _\delta \). In fact, it is only the definition of the semantics of newvar\( _\delta \) that seems to require properties \((\Sigma 1)\), \((\Sigma 2)\), and \((\Sigma 3)\); the semantics of the other linguistic constants can be given with \( \text{Set}^{\text{op}} \) as the category of store shapes (thus omitting the second component \( \rho \) of the arrows \( (\phi, \rho) \) of \( \Sigma \)). In this chapter, the following objects are
defined exactly as they were in Chapter VII:

(1) the functor $\text{Newshape}_\delta \in \Sigma \rightarrow \Sigma$, which is such that, for $X \in \text{Ob } \Sigma$,

$$\text{Newshape}_\delta \ X = X \times \text{Val } \delta,$$

(2) the functions

$$\text{enter}_\delta \ X \in X \rightarrow X \times \text{Val } \delta,$$

$$\text{exit}_\delta \ X \in \text{Val } \delta \rightarrow X$$

used upon entering and exiting from blocks (recall that $\text{enter}_\delta \ X \ x = (x, \text{init } \delta)$ and $\text{exit}_\delta \ X \ (x, v) = x$),

and

(3) the natural transformation

$$\text{Exit}_\delta \in \Sigma \rightarrow_{\Sigma} \text{Newshape}_\delta,$$

which is such that for $X \in \text{Ob } \Sigma$,

$$\text{Exit}_\delta \ X = (\text{exit}_\delta \ X, \nu_\delta \ X) \in X \rightarrow_{\Sigma} X \times \text{Val } \delta.$$ 

Furthermore, Lemma 7.3 and Lemma 7.4 are still valid.

However, for continuation semantics, the notion of "canonical local $\delta$-variable for a store shape $X$" needs revision. For $X \in \text{Ob } \Sigma$ let

$$\text{localvar}_\delta \ X \in \text{Mng } \delta-\text{var } (X \times \text{Val } \delta)$$

be

$$\text{localvar}_\delta \ X = (a_\delta, X, e_\delta, X)$$
where \( a_{\delta, \Sigma} \in \text{Val} \delta \xrightarrow{P_{dom}} \text{Mng} \text{ comm} (X \times \text{Val} \delta) \), and

\[ e_{\delta, \Sigma} \in X \times \text{Val} \delta \xrightarrow{P_{dom}} (\text{Val} \delta)_{\bot} \]

are defined below. Let \( e_{\delta, \Sigma} \) be given by

\[ e_{\delta, \Sigma}(x, v) = v \]

where \( x \in X, \ v \in \text{Val} \delta \) (just as in Chapter VII).

For \( v \in \text{Val} \delta \), we must give a natural transformation

\[ a_{\delta, \Sigma} \in \text{hom}^\Sigma (X \times \text{Val} \delta) \times \text{Mng compl} \xrightarrow{\Sigma} \text{Mng compl}. \]

Hence, for a store shape \( Y \in \text{Ob} \Sigma \), let

\[ a_{\delta, \Sigma} \in \text{Y} \xrightarrow{\Sigma} \text{Y} \times (\text{Y} \rightarrow 0) \xrightarrow{P_{dom}} (\text{Y} \rightarrow 0) \]

be given by

\[ a_{\delta, \Sigma} \in \text{Y} \langle(\phi, \rho), k \rangle y = k (\rho (\text{exit}_{\delta} X (\phi y), v) y). \]

The idea behind the definition is that we apply the continuation \( k \) to an altered state derived from \( y \) by assigning \( v \) to the canonical local \( \delta \)-variable. Verification that

\( a_{\delta, \Sigma} \in \text{Y} \) is continuous is routine. We must verify that

\( a_{\delta, \Sigma} \in \text{Y} \) is a natural transformation. Suppose that

\( (\phi', \rho') \in \text{Y} \xrightarrow{\Sigma} \text{Z} \). Consider the diagram

\[
\begin{array}{ccc}
(X \times \text{Val} \delta) \xrightarrow{\Sigma} \text{Y} \times (\text{Y} \rightarrow 0) & \xrightarrow{a_{\delta, \Sigma} \in \text{Y} \times (\text{Y} \rightarrow 0)} & (\text{Y} \rightarrow 0) \\
\text{hom}^\Sigma (X \times \text{Val} \delta) \langle(\phi', \rho') \times (\phi' \Rightarrow 1) & \xrightarrow{\phi' \Rightarrow 1} & (X \times \text{Val} \delta) \xrightarrow{\Sigma} \text{Z} \times (\text{Z} \rightarrow 0) & \xrightarrow{a_{\delta, \Sigma} \in \text{Y} \times (\text{Z} \rightarrow 0)} & (\text{Z} \rightarrow 0)
\end{array}
\]
For \((\phi, \rho) \in X \times \text{Val} \xrightarrow{\delta} Y, k \in Y \xrightarrow{p \text{dom}} 0, z \in Z\), we have

\[
(\langle \phi, \rho \rangle \cong X \times \text{Val} \xrightarrow{\delta} Y, k \in Y \xrightarrow{p \text{dom}} 0, z \in Z, \text{we have}

= a_{\delta, X \times Y} \circ (\phi' \cong (\phi', \rho')) \circ (\langle \phi, \rho \rangle, k \circ \phi') z

= a_{\delta, X \times Y} \circ (\phi' \cong (\phi', \rho')) \circ (\langle \phi, \rho \rangle, k \circ \phi') z

= (k \circ \phi') \circ (\rho \circ \text{exit}_{\delta} X ((\phi' z), v) \circ \phi') z

= (k \circ \phi') \circ (\rho \circ \text{exit}_{\delta} X ((\phi' z), v) \circ \phi') z

= k (\rho' \circ \phi' \circ \text{exit}_{\delta} X ((\phi' z), v) (\phi' z)) z)

= k (\rho' \circ \phi' \circ \text{exit}_{\delta} X ((\phi' z), v) (\phi' z)) z)

= k (\rho' \circ \phi' \circ \text{exit}_{\delta} X ((\phi' z), v) (\phi' z)) z)

= a_{\delta, X \times Y} \circ Y (\langle \phi, \rho \rangle, k \circ \phi') z

= (a_{\delta, X \times Y} \circ Y (\langle \phi, \rho \rangle, k \circ \phi')) z

= (\phi' \Rightarrow l) \circ a_{\delta, X \times Y} \circ Y (\langle \phi, \rho \rangle, k \circ \phi') z

which shows that the diagram commutes, and \(a_{\delta, X \times Y} \circ Y\) is a natural transformation.
Of course, it is necessary to prove a modified version of Lemma 7.5. Recall that \( \langle \phi_\delta, \rho_\delta \rangle \) is another notation for Newshape_\(\delta\) \(\langle \phi, \rho \rangle\) when \(\langle \phi, \rho \rangle \in X \xrightarrow{\Sigma} Y\). Note the use of property (\(\Xi 2\)) in the proof.

Lemma 8.1: Let \(\langle \phi, \rho \rangle \in X \xrightarrow{\Sigma} Y\). Then

\[
\text{Mng } \delta \text{-var} \left( \phi_\delta, \rho_\delta \right) \left( \text{localvar}_\delta X \right) = \text{localvar}_\delta Y.
\]

Proof: Since

\[
\text{Mng } \delta \text{-var} = \text{Mng } \delta \text{-acc} \times \text{Mng } \delta \text{-exp},
\]

\[
\text{localvar}_\delta X = \langle a_\delta, x', e_\delta, X \rangle,
\]

\[
\text{localvar}_\delta Y = \langle a_\delta, y', e_\delta, Y \rangle,
\]

it suffices to show

\[
\text{Mng } \delta \text{-acc} \left( \phi_\delta, \rho_\delta \right) \ a_\delta, x = a_\delta, y', \quad \text{and}
\]

\[
\text{Mng } \delta \text{-exp} \left( \phi_\delta, \rho_\delta \right) \ e_\delta, x = e_\delta, y'.
\]

Let \(v \in \text{Val}_\delta\), \(Z \in \text{Ob } \Sigma\), \(\langle \phi', \rho' \rangle \in Y \times \text{Val } \delta \xrightarrow{\Sigma} Z\), \(k \in Z \xrightarrow{\rho_{\text{dom}}} 0\) and \(z \in Z\). Observe that if we let \(\phi' z = \langle y, w \rangle\) then we can carry out the following computation:

\[
\rho_\delta \left( \text{exit}_\delta X \left( \phi_\delta \left( \phi' z \right) \right), v \right) \left( \phi' z \right)
\]

\[
= \rho_\delta \left( \text{exit}_\delta X \left( \phi_\delta \langle y, w \rangle \right), v \right) \langle y, w \rangle
\]

\[
= \rho_\delta \left( \text{exit}_\delta X \langle \phi y, w \rangle, v \right) \langle y, w \rangle
\]

\[
= \rho_\delta \langle \phi y, v \rangle \langle y, w \rangle
\]

\[
= \langle \rho \langle \phi y \rangle y, v \rangle
\]
\[ = \langle y, v \rangle, \text{ by } (\Sigma 2), \]

\[ = (\text{exit}_\delta Y (\langle y, w \rangle, v) \]

\[ = (\text{exit}_\delta Y (\phi', z), v). \]

Thus,

\[ \text{Mng} \delta \text{-acc} (\phi', \rho') a, \varepsilon X, v Z ((\phi', \rho'), k) z \]

\[ = \text{Mng} \text{ comm} (\phi', \rho') (a, \varepsilon, X, v) Z ((\phi', \rho'), k) z \]

\[ = a, \varepsilon, X, v Z ((\phi', \rho') \circ (\phi', \rho'), k) z, \]

since \(\text{Mng comm} = \text{Mng compl} \Rightarrow \text{Mng compl} \),

\[ = a, \varepsilon, X, v Z ((\phi', \rho') \circ (\phi', \rho') \circ (\phi', \rho'), k) z \]

\[ = k ((D, \varepsilon (\phi' \circ \rho')) \circ (\phi', \rho') \circ (\phi', \rho') \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

\[ = k (D, \varepsilon ((\phi' \circ \rho')) \circ (\phi', \rho') \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

\[ = k (D, \varepsilon (\rho' \circ \rho') \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

\[ = k ((\rho' \circ \rho') \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

\[ = k (\rho' \circ \rho' \circ (\phi', \rho)' \circ (\phi', \rho') \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

\[ = k (\rho' \circ \rho' \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

by the preceding computation,

\[ = a, \varepsilon, X, v Z ((\phi', \rho') \circ (\phi', \rho') \circ (\phi', \rho) (\text{exit}_\delta X ((\phi', \rho', \rho') z), v) z) \]

This demonstrates

\[ \text{Mng} \delta \text{-acc} (\phi', \rho') a, \varepsilon X = a, \varepsilon, Y. \]

The proof of the other equation is very simple and can be found in the proof of Lemma 7.5. []
We are now ready to give the semantics of $\text{newvar}_\delta$, where $\delta \in \mathcal{D}$. The aim is to give

$$\text{semf } \text{newvar}_\delta \in \text{Env } \text{emp}_T \xrightarrow{\mathcal{K}} (\text{Mng } \delta-\text{var} \Rightarrow \text{Mng } \text{comm}) \Rightarrow \text{Mng } \text{comm},$$

and we do this by giving the corresponding

$$\text{nv}_\delta \in (\text{Mng } \delta-\text{var} \Rightarrow \text{Mng } \text{comm}) \times \text{Mng } \text{compl} \xrightarrow{\mathcal{K}} \text{Mng } \text{compl}.$$ 

Hence, for $X \in \text{Ob } \Sigma$, let

$$\text{nv}_\delta X \in (\text{Mng } \delta-\text{var} \Rightarrow \text{Mng } \text{comm}) \times (X = 0) \xrightarrow{\rho_{\text{dom}}} (X = 0)$$

be given by

$$\text{nv}_\delta X (\eta, k)$$

$$= \eta(X \times \text{Val}_\delta)(\text{Exit}_\delta X, \text{localvar}_\delta X)(X \times \text{Val}_\delta)(\sum_{X \times \text{Val}_\delta} k \circ \text{exit}_\delta X)$$

$$\circ \text{enter}_\delta X,$$

where $\eta \in \text{hom}^\Sigma X \times \text{Mng } \delta-\text{var} \xrightarrow{\mathcal{K}} \text{Mng } \text{comm}$ and $k \in X \xrightarrow{\rho_{\text{dom}}} 0$.

Think of $\eta$ as telling how to construct a command out of a $\delta$-variable. Then $\text{nv}_\delta X (\eta, k)$ means enter a new block, execute a command constructed with $\eta$ from the canonical local $\delta$-variable, then continue with $k \circ \text{exit}_\delta X$ (i.e. exit from the block and continue with $k$). It is straightforward to show that $\text{nv}_\delta X$ is continuous. We turn our attention to showing that $\text{nv}_\delta$ is a natural transformation. Suppose $(\phi, \rho) \in X \xrightarrow{\Sigma} Y$.

The following technical lemma is useful.

**Lemma 8.2:** Let $(\phi, \rho) \in X \xrightarrow{\Sigma} Y$ and $k \in X \xrightarrow{\rho_{\text{dom}}} 0$.

Then
\[ \text{Mng compl} (\phi_\delta, \rho_\delta)(k \circ \text{exit}_\delta X) = k \circ \phi \circ \text{exit}_\delta Y. \]

**Proof:** Both sides of the equation are elements of \( Y \times \text{Val } \delta \) \( \overset{\text{Pdom}}{\longrightarrow} 0 \). Suppose \( y \in Y \) and \( v \in \text{Val } \delta \). The desired equality follows from the computation

\[
\begin{align*}
\text{Mng compl} (\phi_\delta, \rho_\delta)(k \circ \text{exit}_\delta X)(y, v) &= (k \circ \text{exit}_\delta X \circ \phi_\delta)(y, v) \\
&= k(\text{exit}_\delta X(\phi_\delta(y, v))) \\
&= k(\text{exit}_\delta X(\phi y, v)) \\
&= k(\phi y) \\
&= k(\phi(\text{exit}_\delta Y(y, v))) \\
&= (k \circ \phi \circ \text{exit}_\delta Y)(y, v).
\end{align*}
\]

We are interested in the commutativity of

\[
\begin{array}{c}
\text{(Mng } \delta\text{-var } \Rightarrow \text{Mng } \text{comm}) \ X \times (X \Rightarrow 0) \overset{\text{Mng comm}}{\longrightarrow} X \Rightarrow 0 \\
\text{(Mng } \delta\text{-var } \Rightarrow \text{Mng comm}) \langle \phi, \rho \rangle \times (\phi \Rightarrow 1) \overset{\phi \Rightarrow 1}{\longrightarrow} \phi \Rightarrow 1 \\
\text{(Mng } \delta\text{-var } \Rightarrow \text{Mng comm}) \ Y \times (Y \Rightarrow 0) \overset{\text{Mng comm}}{\longrightarrow} Y \Rightarrow 0.
\end{array}
\]

Let \( \eta \in \text{hom}^\Sigma X \times \text{Mng } \delta\text{-var} \overset{K}{\longrightarrow} \text{Mng } \text{comm} \), \( k \in X \overset{\text{Pdom}}{\longrightarrow} 0 \) and \( Y \in Y \). Then

\[
\begin{align*}
(\text{Mng } \delta\text{-var } \Rightarrow \text{Mng } \text{comm}) (\phi, \rho) (\phi \Rightarrow 1))(\eta, k) \ Y &= \text{Mng compl} (\phi_\delta, \rho_\delta)(k \circ \text{exit}_\delta Y) \\
&= \text{Mng compl} (\phi_\delta, \rho_\delta)(k \circ \phi \circ \text{exit}_\delta Y).
\end{align*}
\]
= $\text{Mng}(\delta\text{-var=Mng \ comm})(\phi, \rho) \eta (Y \times \text{Val} \ \delta)(\text{Exit}_\delta X, \ \text{localvar}_\delta X)\\
(Y \times \text{Val} \ \delta)(1^\cap_{Y \times \text{Val} \ \delta}, k \circ \phi \circ \text{exit}_\delta Y)(\text{enter}_\delta Y \ y)\\
= \eta (Y \times \text{Val} \ \delta)(\text{Exit}_\delta X, \ \text{localvar}_\delta X)\\
(Y \times \text{Val} \ \delta)(1^\cap_{Y \times \text{Val} \ \delta}, k \circ \phi \circ \text{exit}_\delta Y)(\text{enter}_\delta Y \ y),\\
\text{by the definition of } \Rightarrow \text{ in } |K|,\\
= \eta (Y \times \text{Val} \ \delta)(\langle \phi, \rho \rangle \circ \text{Exit}_\delta X, \ \text{localvar}_\delta X)\\
(Y \times \text{Val} \ \delta)(1^\cap_{Y \times \text{Val} \ \delta}, k \circ \phi \circ \text{exit}_\delta Y)(\text{enter}_\delta Y \ y),\\
\text{by Lemma 7.4},\\
= \eta (Y \times \text{Val} \ \delta)\\
((\text{hom}^\cap X \langle \phi, \rho \rangle \times \text{Mng} \ \delta\text{-var})(\phi, \rho))(\text{Exit}_\delta X, \ \text{localvar}_\delta X)\\
(Y \times \text{Val} \ \delta)(1^\cap_{Y \times \text{Val} \ \delta}, k \circ \phi \circ \text{exit}_\delta Y)(\text{enter}_\delta Y \ y),\\
\text{by Lemma 8.1},\\
= \text{Mng \ comm}(\phi, \rho)(\eta(X \times \text{Val} \ \delta)(\text{Exit}_\delta X, \ \text{localvar}_\delta X))\\
(Y \times \text{Val} \ \delta)(1^\cap_{Y \times \text{Val} \ \delta}, k \circ \phi \circ \text{exit}_\delta Y)(\text{enter}_\delta Y \ y),\\
\text{by the naturality of } \eta,\\
= \eta (X \times \text{Val} \ \delta)(\text{Exit}_\delta X, \ \text{localvar}_\delta X)(Y \times \text{Val} \ \delta)\\
\langle \langle \phi, \rho \rangle, k \circ \phi \circ \text{exit}_\delta Y \rangle(\text{enter}_\delta Y \ y),\\
because \text{Mng \ comm} = \text{Mng \ compl} \Rightarrow \text{Mng \ compl},\\
= \eta (X \times \text{Val} \ \delta)(\text{Exit}_\delta X, \ \text{localvar}_\delta X)(Y \times \text{Val} \ \delta)\\
((\text{hom}^\cap(X \times \text{Val} \ \delta)(\phi, \rho) \times \text{Mng \ compl})(\phi, \rho))(1^\cap_{X \times \text{Val} \ \delta}, k \circ \text{exit}_\delta X)\\
(\text{enter}_\delta Y \ y), \text{ by Lemma 8.2},
= Mg compl (φδ, ρδ)
(η(X×Valδ)(ExitδX, localvarδX)(X×Valδ)(lX×Valδ, k • exitδX))
(enterδ Y y),

by the naturality of η(X×Valδ)(ExitδX, localvarδX),
= (η(X×Valδ)(ExitδX, localvarδX)(X×Valδ)(lX×Valδ, k • exitδX) • φδ)
(enterδ Y y)
= η(X×Valδ)(ExitδX, localvarδX)(X×Valδ)(lX×Valδ, k • exitδX)
(φδ(enterδ Y y))
= η(X×Valδ)(ExitδX, localvarδX)(X×Valδ)(lX×Valδ, k • exitδX)
(enterδ X (φ y)),
because (φδ(enterδ Y y))
= φδ(y, init δ) = (φ y, init δ) = enterδ X(φ y),
= nvδ X (η, k) (φ y)
= (nvδ X (η, k) • φ) y
= (φ ⇒ l) (nvδ X (η, k) y
= ((φ ⇒ l) • nvδ X) (η, k) y,

and this shows the diagram commutes. Hence nvδ is a natural transformation.

The semantics of halt are provided by

\[ \text{semf} \text{ halt} \in \text{Env emp}_T \xrightarrow{|K|} \text{Mng compl} \]

We assume there is some element of 0 denoting normal program termination; call it endexecute ε 0. Then for a store shape X, let
semf halt \( X \in \text{Env} \) \( \text{emp}_T \xrightarrow{P_{\text{dom}}} (X = 0) \)

be given by

\[
\text{semf halt } X \left( \_ \right) = \text{endexecute}, \text{ for all } x \in X.
\]

It is trivial to see that semf halt is a natural transformation.

(We remark that some other kinds of linguistic constants, such as output statements, must be given if we are to have programs that do more than either print a program termination message or run on indefinitely without giving output. See Reynolds [7]. We cannot delve more deeply into these matters without exploring the story of 0.)

The last linguistic constant whose semantics we must give is goto. We want

\[
\text{semf goto } \in \text{Env} \xrightarrow{_{K}} \text{Mng } \text{compl } \Rightarrow \text{Mng comm};
\]

therefore we will give the corresponding natural transformation

\[
\text{jump } \in \text{Mng compl } \times \text{Mng compl } \xrightarrow{_{K}} \text{Mng compl}.\]

Just let jump be the projection onto the first component. Thus goto interrupts the flow of control by sending control to its argument, rather than following the "normal" flow (represented by the second component of the domain of jump). After newvar, the semantics of goto are mercifully brief.
BIBLIOGRAPHY


BIOGRAPHICAL DATA

Name: Frank Joseph Oles

Date and Place of Birth: December 15, 1946
Kronberg, Germany

Elementary School: Maplewood School
Sylvania, Ohio

Stranahan School
Sylvania, Ohio

Grove Patterson School
Toledo, Ohio

Monac School
Toledo, Ohio
Graduated 1957

Junior High School: Washington Junior High School
Toledo, Ohio

Jefferson Junior High School
Toledo, Ohio
Graduated 1960

High School: Whitmer Senior High School
Toledo, Ohio
Graduated 1964

College: Case Western Reserve University
Cleveland, Ohio
B.S. 1968

Graduate School: Cornell University
Ithaca, New York
M.S. 1970
Syracuse University
Syracuse, New York