Language theory of lambda-calculus with recursive types

Citation for published version (APA):

Document status and date:
Published: 01/01/1988

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Language Theory of a λ-calculus

with Recursive Types

by

H.M.M. ten Eikelder

R.H. Mak

88/14

june 1988
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author or the editor.
0. INTRODUCTION

It is not uncommon to design a programming language by regarding the kind of computations one would like to perform and to decide on a style of notation. Thus one arrives at a syntactic definition of the language which in general contains a large number of constructs and which, for the purpose of expressing ones computations, is usually very satisfying. However, when it comes to assigning a precise meaning to the syntactic constructs thus arrived at, the problems soon become tremendous. Therefore it seems more appropriate to investigate what the proper mathematical abstractions are to model ones computations with and to see in which way they should be manipulated. Thus a carefully chosen (preferably small) number of semantic constructs should dictate the basic syntactic ingredients of a kernel language. Ease of programming can be obtained by adding an additional layer of syntactic sugar to this kernel language. Since the latter is defined in terms of the basic syntactic constructs, it is not hard to define its semantics. Our ultimate goal is to design a language along these lines. Our interest is not so much in the resulting language, however, but rather in the design process itself. As the kernel for our language we have opted for the lambda-calculus, because of its simple nature, extended with a rich type structure, that should allow for instance polymorphism and recursively defined types. There are several approaches known in the literature such as languages with implicit types like ML [HMcQM86] or languages with explicit types as described in [Re85]. In this report we make a start towards the latter in the sense that the language we define does contain recursive types and what is known as a polymorphic let-construct. It does not contain, however, expressions which are 'type-abstractions'. We have chosen this cautious approach, since the semantics of second order lambda calculi with recursive types is not yet well understood, although various results are known [McQPS86,McC79,Me86,Mi71]. Therefore we study this relatively simple case in great detail before we turn our attention towards 'full' polymorphism. Moreover, we have included both strict and non-strict versions of our type constructors. Investigation of their semantic properties will enable us to make the proper choice in a latter stage when we design the actual language.
CHAPTER 0

The structure of this report is as follows. In chapter 1 the language is given and the meaning of its constructs is briefly explained. A comprehensive and formal semantics is given in chapters 4 and 5 for the type expressions and expressions proper respectively. In chapter 2 a type deduction system is given that enables us to keep the type information within expressions to a minimum. Chapter 3 states a set of reduction rules whose soundness is proven in chapter 6. These rules can be viewed as an operational semantics of our language. Finally, in chapter 7, it is shown that a typed version of the Curry fixed point combinator [Ba81,HiSe86] can be defined in the language.
1. SYNTAX OF TYPE EXPRESSIONS AND EXPRESSIONS

The language we consider consists of expressions that contain type information. Its formal syntax is given by two kinds of expressions, type expressions and expressions proper. Let $T\text{var}$ be a countable infinite set of variables. Elements of $T\text{var}$ will be called type variables. Type expressions are generated by the following rules.

T1. $\text{Texp} ::= \Omega$
T2. $\text{Texp} ::= T\text{var}$
T3. $\text{Texp} ::= \uparrow\text{Texp}$
T4.1. $\text{Texp} ::= (\text{Texp} + \text{Texp})$
T4.2. $\text{Texp} ::= (\text{Texp} \circ \text{Texp})$
T5.1. $\text{Texp} ::= (\text{Texp} \times \text{Texp})$
T5.2. $\text{Texp} ::= (\text{Texp} \circ \text{Texp})$
T6.1. $\text{Texp} ::= (\text{Texp} \rightarrow \text{Texp})$
T6.2. $\text{Texp} ::= (\text{Texp} \ominus \text{Texp})$
T7. $\text{Texp} ::= v(A \text{Tvar} | \text{Texp})$

A formal semantics, which associates a domain (c.p.o.) to every type expression, will be defined in section 4. We now give an informal description of the domains corresponding to type expressions generated by T1 - T7. The type expression $\Omega$ corresponds to the one point domain. The symbol $\uparrow$ is used to denote lifting of the domain, i.e. appending a fresh bottom element. Further $+, \times, \rightarrow$ correspond to the disjoint sum, cartesian product and function space domain constructors, whereas $\oplus, \otimes, \ominus$ correspond to their strict versions, i.e. the coalesced sum, smash product and space of strict functions. A type expression of the form $v(A t | t e)$ describes a recursively defined type. For instance the type expression $v(A t | (t + t))$ corresponds to a domain $D$ such that $D$ is isomorphic to the disjoint sum of $D$ and $D$; the type expression $v(A t | (\uparrow \Omega \oplus t))$ describes the flat domain of natural numbers. Whether an actual programming language should contain all the type constructors above remains to be seen. However, it is precisely the intention of this paper to investigate the properties of
the various constructs in order to allow a deliberate choice.

Let \( \text{Var} \) be a countable infinite set of variables such that \( \text{Var} \cap \text{Tvar} = \emptyset \). The syntax of expressions is given by the following rules.

\[
\begin{align*}
\text{E1.} & \quad \text{Exp} ::= (\text{btm} | \text{Exp})^* \\
\text{E2.} & \quad \text{Exp} ::= \text{Var}^* \\
\text{E3.1.} & \quad \text{Exp} ::= (\text{up} \text{Exp})^* \\
\text{E3.2.} & \quad \text{Exp} ::= (\text{down} \text{Exp})^* \\
\text{E4.1.1.} & \quad \text{Exp} ::= (\text{inl} \text{Exp} | \text{Exp})^* \\
\text{E4.1.2.} & \quad \text{Exp} ::= (\text{inr} \text{Exp} | \text{Exp})^* \\
\text{E4.1.3.} & \quad \text{Exp} ::= (\text{sum} \text{Exp} \text{Exp})^* \\
\text{E4.2.1.} & \quad \text{Exp} ::= (\text{inls} \text{Exp} | \text{Exp})^* \\
\text{E4.2.2.} & \quad \text{Exp} ::= (\text{inrs} \text{Exp} | \text{Exp})^* \\
\text{E4.2.3.} & \quad \text{Exp} ::= (\text{sums} \text{Exp} \text{Exp})^* \\
\text{E5.1.1.} & \quad \text{Exp} ::= (\text{prol} \text{Exp})^* \\
\text{E5.1.2.} & \quad \text{Exp} ::= (\text{pror} \text{Exp})^* \\
\text{E5.1.3.} & \quad \text{Exp} ::= (\text{prod} \text{Exp} \text{Exp})^* \\
\text{E5.2.1.} & \quad \text{Exp} ::= (\text{prols} \text{Exp})^* \\
\text{E5.2.2.} & \quad \text{Exp} ::= (\text{prors} \text{Exp})^* \\
\text{E5.2.3.} & \quad \text{Exp} ::= (\text{prods} \text{Exp} \text{Exp})^* \\
\text{E6.1.1.} & \quad \text{Exp} ::= (\lambda \text{Var}:\text{Exp} | \text{Exp})^* \\
\text{E6.1.2.} & \quad \text{Exp} ::= (\text{appl} \text{Exp} \text{Exp})^* \\
\text{E6.2.1.} & \quad \text{Exp} ::= (\lambda s \text{Var}:\text{Exp} | \text{Exp})^* \\
\text{E6.2.2.} & \quad \text{Exp} ::= (\text{appls} \text{Exp} \text{Exp})^* \\
\text{E7.1.} & \quad \text{Exp} ::= (\text{intro} s (\lambda \text{Tvar} | \text{Exp}) | \text{Exp})^* \\
\text{E7.2.} & \quad \text{Exp} ::= (\text{elim} s (\lambda \text{Tvar} | \text{Exp}) | \text{Exp})^* \\
\text{E8.} & \quad \text{Exp} ::= (\lambda \text{Tvar} | \text{Exp}) \text{Exp}^* 
\end{align*}
\]
In chapter 2 we give a type deduction system that defines the well typed expressions. Furthermore it will be shown that every well typed expression has exactly one type (up to $\alpha$–conversion). In chapter 5 we define the semantics of a well typed expression and show that the value of an expression is an element of the domain corresponding to its type. An operational semantics in terms of reduction rules is given in section 3.

In the rest of this chapter we give an informal description of the expressions introduced above. Let $te$ be a type expression. The expression $(\text{btm} | te)$ stands for a nonterminating computation which does not yield any information. The expressions generated by E3 are used in connection with the lifting of domains. In particular the $(\text{up} e)$ construct is used to postpone reductions inside the expression $e$ (see also chapter 3). The expressions defined by E4.1 are related to the disjoint sum of domains: $(\text{inl} e | te)$ and $(\text{inr} te | e)$ denote the injection of $e$ in the left respectively right part of a sum domain. If $e_1$ and $e_2$ denote two functions with the same range, then $(\text{sum} e_1 e_2)$ denotes a function whose domain is the disjoint sum of the domains of $e_1$ and $e_2$ and whose range is the common range of $e_1$ and $e_2$. The expressions defined by E4.2 are the strict versions of those given in E4.1, they correspond to the strict sum of domains $(\oplus)$. E5.1 generates expressions which are related to the product of domains. The first two rules correspond to the left and right projection, whereas E5.1.3 corresponds to the pair construction. Again E5.2 gives the strict versions. E6.1 (and E6.2) describe (strict) lambda abstraction and application. To understand E7 consider a recursively defined type expression, for instance $v(\Delta t | t + t)$. The domain $D$ which will be associated to this type expression (see chapter 4) is isomorphic to the disjoint sum of $D$ and $D$. The two expressions given by E7 are the syntactic representants of these kinds of isomorphism and its inverse. Finally E8 gives the possibility of building a context of type variables which are bound to type expressions.

Next we introduce some notations which will be used frequently in this report. The mapping $\text{FV} : \text{Exp} \rightarrow \text{Var}$ yields the free variables of an expression. The mapping $\text{FTV} : \text{Exp} \cup \text{Texp} \rightarrow \text{Tvar}$
CHAPTER 1

gives the free type variables of an expression or a type expression. Recursive definitions of FV and FTV can easily be given, but we shall not do so here. In the sequel we shall encounter three kinds of substitution. The substitution of type expressions for type variables can be performed in type expressions and in expressions. The substitution of expressions for variables can only take place in expressions. Apart from the case of (type) expressions with bounded (type) variables the definition of substitution is straightforward. In case of substitution for a type variable in a (type) expression with a bounded type variable or substitution for a variable in an expression with a bounded variable name clashes may occur. In that case the bounded (type) variable is always replaced by the first appropriate free (type) variable. We list the instances where this happens. Let \( s, t \in \text{Tvar} \), \( x, y \in \text{Var} \), \( t_1, t_2 \in \text{Texp} \) and \( e, e_1, e_2 \in \text{Exp} \). Then

- \((\forall t_1 \mid t_2)^{s}_{t_2} = \forall u \mid (t_1^{t_1})^{s}_{u \mid t_2} \),
  where \( u \) is the first type variable such that \( u \neq s \) and \( u \in \text{FTV}(t_1) \cup \text{FTV}(t_2) \).

- \((\lambda x : t_1 \mid e_1)^{y}_{t_2} = (\lambda z : t_1 \mid (e_1^x)^{y}_{z} \mid t_2) \),
  where \( z \) is the first variable such that \( z \neq y \) and \( z \in \text{FV}(e_1) \cup \text{FV}(e_2) \).

- \((\Lambda t_1 \mid e)^{s}_{t_2} = (\Lambda u \mid (e^t)^{s}_{u \mid t_2} \mid t_1^s_{t_2}) \),
  where \( u \) is the first type variable such that \( u \neq s \) and \( u \in \text{FTV}(e) \cup \text{FTV}(t_1) \cup \text{FTV}(t_2) \).

Here \( t_1 \) is the type expression which will be associated to \( e \) by the type inference system given in the next chapter (hence substitution is only defined for well-typed expressions).

Note that our definition of substitution implies that bound variables will also be renamed in cases where this is in fact not necessary. The reason for choosing this definition, instead of a more usual one which considers several cases [Ba81], is to reduce the case analysis in the proofs further on. Finally we mention that the symbol \( \equiv \) will be used to denote the syntactic equality of (type) expressions, whereas \( \equiv_{\alpha} \) will be used for the equality of (type) expressions up to renaming of the bound variables (\( \alpha \)-conversion).
2. TYPE INFERENCE

2.1. Introduction.

In this chapter we demonstrate that the kernel language introduced in the previous chapter is an explicitly typed language in the sense of Reynolds[Re85]. That is, given an expression and a sequence of assumptions regarding the free variables and free type variables occurring in that expression it is possible to assert at most one type for that expression. By a type we mean a class of type expressions that are equal up to $\alpha$–conversion. In chapter 4 it is shown that all type expressions in such a class denote the same domain.

2.2. Formal type inference system.

Formula's of the type inference system will be called typings and they are constructed according to the following grammar rules:

I1. Typing ::= Assumptions \(\Rightarrow\) Consequences ·
I2.1. Assumptions ::= ·
I2.2. Assumptions ::= Assumption Rest ·
I3.1. Assumption ::= Type assignment ·
I3.2. Assumption ::= Tvar ·
I4.1. Rest ::= ·
I4.2. Rest ::= ; Assumption Rest ·
I5.1. Consequences ::= Consequences , Consequences ·
I5.2. Consequences ::= Type assertion ·
I5.3. Consequences ::= Texp ·
I6. Type assignment ::= Var : Texp ·
I7. Type assertion ::= Exp : Texp ·
CHAPTER 2

For instance, the typing \( t;\text{x:t} \vdash (\text{inl} \ x \mid t) : t + t \) states that under the assumptions that (there exists a context in which) first of all a type \( t \) is introduced and secondly a variable \( x \) of type \( t \), one may assert that the expression \( (\text{inl} \ x \mid t) \) is of type \( t + t \). As usual we prefix a typing with the symbol \( \vdash \) to indicate that it is derivable.

Let \( A \in \text{Assumptions} \). The set \( \text{FTV}(A) \) of free type variables of \( A \) is the set of type variables that occur as subassumptions in \( A \) (cf. I3.2). Hence for \( x:t \) an assumption \( t \notin \text{FTV}(x:t) \)!

The set \( \text{FV}(A) \) of free variables of \( A \) is the set of variables that occur in any left–hand side of any type assignment in \( A \) (cf. I3.1 and I6).

Let \( C \in \text{Consequences} \). The set \( \text{FTV}(C) \) is the set of type variables occurring free in any expression or type expression contained in \( C \) (cf. I5.3 and I7). In particular \( \text{FTV}(e;te) = \text{FTV}(e) \cup \text{FTV}(te) \). Hence if \( x:t \) is a consequence then \( t \in \text{FTV}(x:t) \) (cf. above)! Similarly, \( \text{FV}(C) \) is the set of free variables occurring in any expression contained in \( C \) (cf. I7). In particular \( \text{FV}(e;te) = \text{FV}(e) \).

Let \( A, A_1, A_2 \in \text{Assumptions} \); \( C_1, C_2 \in \text{Consequences} \); \( t \in \text{Tvar} \); \( tx, te, te_1, te_2 \in \text{Texp} \); \( x \in \text{Var} \) and \( e, e_1, e_2, f, f_1, f_2 \in \text{Exp} \). Then the inference rules for type deduction are:

<table>
<thead>
<tr>
<th>Rule</th>
<th>premises</th>
<th>conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR1.</td>
<td>( A \vdash \Omega )</td>
<td>( A \vdash t )</td>
</tr>
<tr>
<td>TR2.</td>
<td>( A; t; A_2 \vdash t )</td>
<td>( A \vdash \uparrow \text{te} )</td>
</tr>
<tr>
<td>TR3.</td>
<td>( A \vdash \text{te} )</td>
<td>( A \vdash \text{te} )</td>
</tr>
<tr>
<td>TR4.</td>
<td>( A \vdash \text{te}_1, \text{te}_2 )</td>
<td>( A \vdash \text{te}_1, \text{te}_2 )</td>
</tr>
<tr>
<td>TR5.</td>
<td>( A \vdash \text{te}_1 + \text{te}_2 )</td>
<td>( A \vdash \text{te}_1 \times \text{te}_2 )</td>
</tr>
<tr>
<td>TR6.</td>
<td>( A \vdash \text{te}_1 \ominus \text{te}_2 )</td>
<td>( A \vdash \text{te}_1 \ominus \text{te}_2 )</td>
</tr>
<tr>
<td>TR7.</td>
<td>( A; t \vdash \text{te} )</td>
<td>( A \vdash v(A; t</td>
</tr>
<tr>
<td>ER1.1.</td>
<td>( A \vdash \text{te} )</td>
<td>( A \vdash (\text{htm}</td>
</tr>
</tbody>
</table>
ER2. \( A_1 \triangleright tx \), provided \( x \in \text{FV}(A_2) \) and \( \text{FTV}(tx) \cap \text{FTV}(A_2) = \emptyset \)

\[ A_1; x : tx; A_2 \triangleright x : tx \]

ER3.1. \( A \triangleright c : te \),

\( A \triangleright (\text{up } c) : \uparrow te \)

ER3.2. \( A \triangleright e : \uparrow te \)

\( A \triangleright (\text{down } e) : te \)

ER4.1. \( A \triangleright e_1 : te_1, te_2 \)

\( A \triangleright (\text{inl } e_1 \mid te_2) : te_1 + te_2 \)

\( A \triangleright (\text{inls } e_1 \mid te_2) : te_1 \otimes te_2 \)

ER4.2. \( A \triangleright te_1, e_2 : te_2 \)

\( A \triangleright (\text{inr } te_1 \mid e_2) : te_1 + te_2 \)

\( A \triangleright (\text{inrs } te_1 \mid e_2) : te_1 \otimes te_2 \)

ER4.3.1 \( A \triangleright f_1 : te_1 \rightarrow te, f_2 : te_2 \rightarrow te \)

\( A \triangleright (\text{sums } f_1 f_2) : (te_1 + te_2) \rightarrow te \)

ER4.3.2 \( A \triangleright f_1 : te_1 \uplus te, f_2 : te_2 \uplus te \)

\( A \triangleright (\text{sums } f_1 f_2) : (te_1 \uplus te_2) \uplus te \)

ER5.1. \( A \triangleright c : te_1 \times te_2 \)

\( A \triangleright (\text{prol } c) : te_1 \)

\( A \triangleright (\text{pror } c) : te_2 \)

ER5.2. \( A \triangleright e : te_1 \otimes te_2 \)

\( A \triangleright (\text{prols } e) : te_1 \)

\( A \triangleright (\text{prors } e) : te_2 \)

ER5.3. \( A \triangleright e_1 : te_1, e_2 : te_2 \)

\( A \triangleright (\text{prod } e_1 e_2) : te_1 \times te_2 \)

\( A \triangleright (\text{prods } e_1 e_2) : te_1 \otimes te_2 \)

ER6.1. \( A; x : tx \triangleright e : te \)

\( A \triangleright (\lambda x : tx \mid e) : tx \rightarrow te \)

\( A \triangleright (\lambda s x : tx \mid e) : tx \uplus te \)

ER6.2. \( A \triangleright f : te \rightarrow te_1, e : te \)

\( A \triangleright (\text{appl } f e) : te_1 \)

ER6.3. \( A \triangleright f : te \uplus te_1, e : te \)

\( A \triangleright (\text{appls } f e) : te_1 \)
CHAPTER 2

ER7.1. \[ A \vdash e : \nu(\lambda t \mid te) \]
\[ A \vdash (\text{intro } \nu(\lambda t \mid te) \mid e) : te^t_{\nu(\lambda t \mid te)} \]

ER7.2. \[ A \vdash e : te^t_{\nu(\lambda t \mid te)}, \nu(\lambda t \mid te) \]
\[ A \vdash (\text{elim } \nu(\lambda t \mid te) \mid e) : \nu(\lambda t \mid te) \]

ER8. \[ A \vdash te_1 \]
\[ A; t \vdash e : te \]
\[ A \vdash (\lambda t \mid e)te_1 : te^t_{te_1} \]

ER9. \[ A \vdash e : te_1 \]
\[ A \vdash e : te_2 \]
\[ A \vdash C_1 \]
\[ A \vdash C_2 \]
\[ A \vdash C_1, C_2 \]
\[ A \vdash C_1, C_2 \]

provided \( te_1 \equiv_\alpha te_2 \)

ER10.1. \[ A \vdash C_1 \]
\[ A \vdash C_2 \]
\[ A \vdash C_1, C_2 \]

ER10.2. \[ A \vdash C_1, C_2 \]
\[ A \vdash C_1 \]
\[ A \vdash C_2 \]

Notice that to each T- and E-rule of chapter 1 there corresponds exactly one inference rule. The additional rule ER9 signifies that we are only interested in type expressions up to \( \alpha \)-conversion. The reason for this is that type expressions that are equal up to \( \alpha \)-conversion denote the same domain. Rules ER10 are not essential. They merely allow us the notational convenience of typings containing more than one consequence. Therefore we shall leave applications of these rules implicit in the derivation of typings.

Most proofs given below rely on the fact that given a typing we are able to determine the last inference rule of its derivation. In the absence of rule ER9 this last rule would be uniquely identifiable from the structure of the expression. Derivations of typings in which the expressions contain bound type variables, however, can always end with one or more applications of rule ER9. In order to avoid these trivial but cumbersome details we assume in all proofs, and without loss of generality, that no derivation ends with an application of rule ER9.
2.3. Explicit typing.

Our type inference system has been designed to ensure that under any given sequence of assumptions each expression has at most one type, which is, if it exists, derivable from the types of its constituting parts (Recall that a type is an equivalence class of type expressions under $\alpha$-conversion.) In Reynolds[Re85] this property is called explicit typing.

**Theorem 2.3.1. [Explicit typing theorem]**

Let $A \in \text{Assumptions}, t_{e1}, t_{e2} \in \text{Texp}$ and $e \in \text{Exp}$. If both $\vdash A \triangleright e : t_{e1}$ and $\vdash A \triangleright e : t_{e2}$ then $t_{e1} =_\alpha t_{e2}$.

**Sketch of proof.** By induction on the structure of expression $e$. Note that to each of the rules E1.1 thru E8 to construct expressions there corresponds exactly one inference rule that enables us to assert a type for the expressions produced by that rule. Therefore the induction is straightforward.

$\square$

As stated above it is necessary to provide expressions with a certain amount of type information to obtain an explicitly typed language. The need for additional type information in expressions produced by E4.1, E4.2 and E6.1 is rather obvious (see p.e. [Re85]). The reader may wonder, however, about the necessity of the type information contained in rules E7.1 and E7.2. Therefore let us assume, for the sake of the argument, that rule E7.2 is simplified to

\[ \text{Exp} ::= (\text{elim } \text{Exp}) \]

and that inference rule ER7.2 is accordingly modified to

\[ A \triangleright e : \text{t}_{e\leftarrow\mu(A \text{ t} \mid \text{te})}, \nu(A \text{ t} \mid \text{te}) \]

\[ A \triangleright (\text{elim } e) : \nu(A \text{ t} \mid \text{te}) \]

\[ (\leftarrow) \]

Let $A \in \text{Assumptions}$ be such that $\vdash A \triangleright e : \text{t}_{e\leftarrow\mu(A \text{ t} \mid \text{t})}, \nu(A \text{ t} \mid \text{t})$. Then we may assert on
account of \((*)\) that \((\text{elim } e)\) is of type \(v(\Lambda t|t)\). However, since \(t_{v(\Lambda t|t)}^\alpha\equiv_{v(\Lambda s|v(\Lambda t|t))}^A\) we may apply rule ER9 before applying \((*)\) and assert that \((\text{elim } e)\) has type \(v(\Lambda s|v(\Lambda t|t))\) as well. Therefore the type information \(v(\Lambda \text{Var}|\text{Exp})\) is absolutely essential in rule E7.2 to obtain explicit typing. For reasons of symmetry the same type information has been added to rule E7.1, although one can show that explicit typing can be obtained without it.

Given an assumption \(A\) we define the set \(\text{WTV}(A)\) (\(\text{WTE}(A)\)) of well–typed variables (expressions) under \(A\) by

\[
\text{WTV}(A) = \{ x \in \text{Var} \mid (\exists \, \text{te} \in \text{Texp} \mid \vdash A \triangleright x : \text{te}) \} \tag{2.3.2}
\]

\[
\text{WTE}(A) = \{ e \in \text{Exp} \mid (\exists \, \text{te} \in \text{Texp} \mid \vdash A \triangleright e : \text{te}) \} \tag{2.3.3}
\]

On account of the explicit typing theorem one can also define for each assumption \(A\) a function \(\tau_A\) that assigns to each expression \(e \in \text{WTE}(A)\) an arbitrary, but fixed, type expression \(\text{te}\) such that \(\vdash A \triangleright e : \text{te}\). We shall take care that whenever \(\tau_A\) is used, the particular \(\text{te}\) chosen for \(\tau_A(e)\) is irrelevant, i.e. may be replaced by any type expression \(\text{te1}\) such that \(\text{te1} \equiv_{\alpha} \text{te}\).

**2.4. Elementary properties.**

Before we state the fundamental properties of our type inference system, viz. inference rules for substitution and \(\alpha\)–conversion, we first list some elementary properties of typings.

**Property 2.4.1.** (Introduction of type variables)

For \(A \in \text{Assumptions}\) and \(\text{te} \in \text{Texp}\):

\(\vdash A \triangleright \text{te} \iff \text{FTV}(\text{te}) \subseteq \text{FTV}(A)\)

\(\Box\)

This property expresses that all free type variables of a type expression should be properly introduced.
Property 2.4.2. [Additional inference rules]

The following additional inference rules are derivable from the ones given in section 2.2:

Rules to extend assumptions

ER11.1. \[ \begin{array}{c} A \vdash C \\ A; t \vdash C \end{array} \] provided \( t \in \text{FTV}(C) \)

ER11.2. \[ \begin{array}{c} A \vdash tx, C \\ A; x: tx \vdash C \end{array} \] provided \( x \in \text{FV}(C) \)

Rules to reorder assumptions

ER12.1. \[ \begin{array}{c} A_1; s; t; A_2 \vdash C \\ A_1; t; s; A_2 \vdash C \end{array} \]

ER12.2. \[ \begin{array}{c} A_1; x: tx; y; ty; A_2 \vdash C \\ A_1; y; ty; x: tx; A_2 \vdash C \end{array} \] provided \( x \neq y \lor ty \equiv_{\alpha} ty \)

ER12.3. \[ \begin{array}{c} A_1; x: tx; t; A_2 \vdash C \\ A_1; t; x: tx; A_2 \vdash C \end{array} \]

ER12.4. \[ \begin{array}{c} A_1; t; x; tx; A_2 \vdash C \\ A_1; x; tx; t; A_2 \vdash C \end{array} \] provided \( t \in \text{FTV}(tx) \)

2.5. Substitution and \( \alpha \)-conversion.

As indicated in chapter 1 three kinds of substitution can be performed. For each kind we present a corresponding inference rule. Likewise three kinds of \( \alpha \)-conversion can be performed. Three additional inference rules state that each kind of \( \alpha \)-conversion leaves the types of expressions invariant. In chapters 4 and 5 we shall demonstrate that \( \alpha \)-conversion neither changes the meaning of type expressions nor the meaning of expressions.
Theorem 2.5.1. [Substitution of type expressions for type variables in type expressions]

Let \( A_1, A_2 \in \text{Assumptions} \); \( t \in \text{Tvar} \) and \( t e, t e 1 \in \text{Texp} \). Then the following inference rule can be derived.

\[
\frac{A_1 \vdash t e \quad A_1; t ; A_2 \vdash t e \quad A_1; A_2 t \vdash t e t \quad \text{provided } t \notin \text{FTV}(A_2)}{A_1; A_2 t e 1 \vdash t e t e 1}
\]

Proof. By induction on the structure of type expression \( t e \). All other cases being trivial we only consider the case \( t e = \nu (\Lambda s \mid t f) \).

Assume

1. \( \vdash A_1 \vdash t e 1 \) \( \tag{**} \)
2. \( \vdash A_1; t; A_2 \vdash t e \) \( \tag{**} \)
3. \( t \notin \text{FTV}(A_2) \) \( \tag{***} \)

1. Let \( t e = \nu (\Lambda s \mid t f) \)
2. \( \vdash A_1; t; A_2; s \vdash t f \) \( \tag{**}, \text{TR7} \)
3. Let \( u \) be the first type variable such that
   \( u \notin t \land u \notin \text{FTV}(t f) \land u \notin \text{FTV}(t e 1) \)
4. \( \vdash A_1; t; A_2; s; u \vdash t f \) \( \tag{2}, (3), \text{ER11.1} \)
5. \( \vdash A_1; t; A_2; u \vdash u \) \( \tag{TR2} \)
6. \( \vdash A_1; t; A_2; u; s \vdash t f \) \( \tag{4}, \text{ER12.1} \)
7. \( \vdash A_1; t; A_2; u \vdash t f s \\ u \) \( \tag{5}, (6), \text{IH} \)
8. \( t \notin \text{FTV}(A_2; u) = \text{FTV}(A_2) \cup \{u\} \) \( \tag{***}, (3) \)
9. \( \vdash A_1; (A_2; u) t \vdash (t f s) t u \) \( \tag{*,(7),(8),IH} \)
10. \( \vdash A_1; A_2 t e 1; u \vdash (t f s) t u \) \( \tag{3}, (9) \)
11. \( \vdash A_1; A_2 t e 1 \vdash \nu (\Lambda u \mid (t f s) t u) \) \( \tag{10}, \text{TR7} \)
12. \( \vdash A_1; A_2 t e 1 \vdash \nu (\Lambda s \mid t f) t e 1 \) \( \tag{3}, (11), \text{subst.} \)
13. \( \vdash A_1; A_2 t e 1 \vdash t e t e 1 \) \( \tag{1}, (12) \)

\( \square \)
Theorem 2.5.2. [Substitution of type expressions for type variables in expressions]

Let $A_1; A_2 \in \text{Assumptions}$; $t \in \text{Tvar}$; $te, te_1 \in \text{Texp}$ and $e \in \text{Exp}$. Then the following inference rule can be derived

$$\vdash A_1 \Rightarrow te_1$$

ER14. $\vdash A_1; t; A_2 \Rightarrow e : te$

provided $\text{FTV}(A_1; t) \cap \text{FTV}(A_2) = \emptyset$

$$\vdash A_1; A_2 t^t_{te_1} \Rightarrow e^t_{te_1} : te^t_{te_1}$$

Proof. By induction on the structure of expression $e$. We prove only a few cases. The remaining cases are trivial.

Assume

$$\vdash A_1 \Rightarrow te_1 \quad (*)$$

$$\vdash A_1; t; A_2 \Rightarrow e : te \quad (**)$$

$$\text{FTV}(A_1; t) \cap \text{FTV}(A_2) = \emptyset \quad (***)$$

1.1. Let $e = x$ and $x \notin \text{FV}(A_2)$, hence $x \in \text{FV}(A_1)$

1.2. Let $A_3, A_4 \in \text{Assumptions}$ be such that

a) $A_1 = A_3; x: te; A_4$

b) $\vdash A_3 \Rightarrow te$

c) $x \notin \text{FV}(A_4; t; A_2)$

d) $\text{FTV}(te) \cap \text{FTV}(A_4; t; A_2) = \emptyset$

1.3. $x \notin \text{FV}(A_4; A_2 t^t_{te_1}) = \text{FV}(A_4; t; A_2)$

(1.2c)

1.4. $\text{FTV}(te) \cap \text{FTV}(A_4; A_2 t^t_{te_1}) = \emptyset$

(***), (1.2d)

1.5. $\vdash A_3; x: te; A_4; A_2 t^t_{te_1} \Rightarrow x : te$

((1.2b), (1.3), (1.4), ER2)

1.6. $t \notin \text{FTV}(te)$

(1.2d)

1.7. $te = \alpha^t_{te_1}$

(1.6)

1.8. $\vdash A_1; A_2 t^t_{te_1} \Rightarrow x : te$

((1.2a), (1.5)

1.9. $\vdash A_1; A_2 t^t_{te_1} \Rightarrow x^t_{te_1} : te^t_{te_1}$

((1.7), (1.8), ER9)

1.10. $\vdash A_1; A_2 t^t_{te_1} \Rightarrow e^t_{te_1} : te^t_{te_1}$

((1.1), (1.9))
2.1. Let \( e = x \) and \( x \in \text{FV}(A_2) \)

2.2. Let \( A_3, A_4 \in \text{Assumptions} \) be such that \([**, ER2]\)

a) \( A_2 = A_3; x : t_1 ; A_4 \)

b) \( \vdash A_1; t_1 ; A_3 \rightarrow t_1 \)

c) \( x \notin \text{FV}(A_4) \)

d) \( \text{FTV}(t_1) \cap \text{FTV}(A_4) = \emptyset \)

2.3. \( a) t \notin \text{FTV}(A_3) \) \[(***)],(2.2a)]

b) \( t \notin \text{FTV}(A) \) \[(***)],(2.2a)]

2.4. \( \vdash A_1; A_3_{t_1} \rightarrow t_1 \)

2.5. \( x \in \text{FV}(A_{4_{t_1}}) = \text{FV}(A_4) \) \[(2.2c)]

2.6. \[\text{FTV}(t_1) \cap \text{FTV}(A_{4_{t_1}}) = \text{FTV}(t_1) \cap \text{FTV}(A_4) = ((\text{FTV}(t_1) \setminus \{t_1\}) \cup \text{FTV}(t_1)) \cap \text{FTV}(A_4) = \text{FTV}(t_1) \cap \text{FTV}(A_4) \subset \text{FTV}(A_4) \cap \text{FTV}(A_4) = \emptyset \] \[(***)],(2.2a)]

2.7. \( \vdash A_1; A_3_{t_1}; x : t_1 ; A_4_{t_1} \rightarrow x : t_{1_{t_1}} \) \[(2.4),(2.5),(2.6), ER2\]

2.8. \( \vdash A_1; A_2_{t_1} \rightarrow x_{t_1} : t_{1_{t_1}} \) \[(2.2a),(2.7), \text{subst}.]\]

2.9. \( \vdash A_1; A_2_{t_1} \rightarrow e_{t_1} : t_{1_{t_1}} \) \[(2.1),(2.8)]\]

3.1. Let \( e \equiv (\lambda y : t_1 | f) \)

3.2. Let \( tf \in \text{Texp} \) be such that \([**, ER6.1]\)

a) \( \vdash A_1; t_1 ; A_2 \rightarrow t_y , tf \)

b) \( \vdash A_1; t_1 ; A_3; y : t_1 \rightarrow f : t_1 \)

c) \( \vdash \text{te} \equiv \alpha _1 ty \rightarrow tf \)

3.3. \( \text{FTV}(A_1; t_1) \cap \text{FTV}(A_2; y : t_1) = \emptyset \) \[(***)\]

3.4. \( \vdash A_1; (A_2; y : t_1)_{t_1} \rightarrow f_{t_1} : t_{1_{t_1}} \) \[(*,3.2b),(3.3), \text{IH}]\
3.5. \( \vdash A_1; A_2^{te_1} \triangleright ty_{te_1}^t, t_{te_1}^t \) \([(*),(**), (3.2a), \text{thm2.5.1}]\)

3.6. \( \vdash A_1; A_2^{te_1} \triangleright y: ty_{te_1}^t : t_{te_1}^t \) \([3.4]\)

3.7. \( \vdash A_1; A_2^{te_1} \triangleright (\lambda y: ty_{te_1}^t \mid f_{te_1}^t) : ty_{te_1}^t \rightarrow t_{te_1}^t \) \([3.5],(3.6), \text{ER6.1}\)

3.8. \( \vdash A_1; A_2^{te_1} \triangleright e_{te_1}^t : t_{te_1}^t \) \([3.1],(3.2c),(3.7)\)

4.1. Let \( e = (\text{elim } v(\Lambda s \mid tf) \mid f) \)

4.2. \( \vdash A_1; A_2 \triangleright f : t_{te_1}^t \nu(\Lambda s \mid tf) \cdot v(\Lambda s \mid tf) \) \([(**), \text{ER7.2}]\)

4.3. \( \vdash A_1; A_2^{te_1} \triangleright f_{te_1}^t : (t_{v(\Lambda s \mid tf)}^t)^{te_1} \) \([(*),(4.2),(**), \text{IH}]\)

4.4. Let \( r \) be the first type variable such that 
\[ r \neq t \land r \in \text{FTV}(tf) \land r \in \text{FTV}(te_1) \]

4.5. \( v(\Lambda s \mid tf)^{te_1}_{te_1} = v(\Lambda r \mid (t_{r_{te_1}}^r)^{te_1}) \) \([4.4, \text{subst}]\)

4.6. \( (t_{v(\Lambda s \mid tf)}^{te_1})^{te_1} = \alpha (t_{v(\Lambda r \mid (t_{r_{te_1}}^r)^{te_1}}^{te_1}) \) \([4.4]\)

4.7. \( \vdash A_1; A_2^{te_1} \triangleright f_{te_1}^t : (t_{v(\Lambda r \mid (t_{r_{te_1}}^r)^{te_1}}^{te_1}) \) \([4.3],(4.6), \text{ER9}]\)

4.8. \( \vdash A_1; A_2^{te_1} \triangleright v(\Lambda s \mid tf)^{te_1}_{te_1} \) \([*,(4.2),(**), \text{thm2.5.1}]\)

4.9. \( \vdash A_1; A_2^{te_1} \triangleright v(\Lambda r \mid (t_{r_{te_1}}^r)^{te_1}) \) \([4.4],(4.8), \text{subst}]\)

4.10. \( \vdash A_1; A_2^{te_1} \triangleright (\text{elim } v(\Lambda r \mid (t_{r_{te_1}}^r)^{te_1}) \mid f_{te_1}^t) : v(\Lambda r \mid (t_{r_{te_1}}^r)^{te_1}) \) \([4.7],(4.9), \text{ER7.2}]\)

4.11. \( \vdash A_1; A_2^{te_1} \triangleright (\text{elim } v(\Lambda s \mid tf) \mid f_{te_1}^t) : v(\Lambda s \mid tf)^{te_1}_{te_1} \) \([4.10], \text{subst}]\)

4.12. \( te = _\alpha v(\Lambda s \mid tf) \) \([**, \text{ER7.2}]\)

4.13. \( \vdash A_1; A_2^{te_1} \triangleright e_{te_1}^t : t_{te_1}^t \) \([4.1],(4.11), (4.12)]\)

5.1. Let \( e = (\Lambda s \mid f)tf_1 \)

5.2. \( \vdash A_1; A_2 \triangleright tf_1 \) \([**, \text{ER8}]\)

5.3. Let \( tf \in \text{Texp} \) be such that 
\[ \begin{align*}
    a) & \vdash A_1; A_2; s \triangleright f:tf \\
    b) & te = _\alpha t_{tf}^s tf_1
\end{align*} \]
5.4. Let \( r \) be the first type variable such that

a) \( r \in \text{FTV}(f:t^{f}) \)

b) \( r \neq t \land r \in \text{FTV}(t_1) \)

5.5. \( \vdash A_1; A_2; s \vdash f : t^{f} \) \hspace{1cm} [(5.3a), (5.4a), ER11.1]

5.6. \( \vdash A_1; A_2; r \vdash f : t^{f} \) \hspace{1cm} [(5.5), ER12.1]

5.7. \( \vdash A_1; A_2; r \vdash r \) \hspace{1cm} [TR2]

5.8. \( \vdash A_1; A_2; r \vdash f : t^{s}_r \) \hspace{1cm} [(5.6), (5.7), IH]

5.9. \( t \in \text{FTV}(A_2;r) \) \hspace{1cm} [(**), (5.4b)]

5.10. \( \vdash A_1; (A_2; r)^{t}_{t_1} \vdash (f^{s}_{t_1} : (t^{s}_{t_1} : t_1)) \) \hspace{1cm} [(*) , (5.8), (5.9), IH]

5.11. \( \vdash A_2; t^{s}_{t_1} \vdash (f^{s}_{t_1} : (t^{s}_{t_1} : t_1)) \) \hspace{1cm} [(5.10)]

5.12. \( \vdash A_2; t^{s}_{t_1} \vdash t_1^{t} \) \hspace{1cm} [(*) , (**), thm2.5.1]

5.13. \( \vdash A_2; t^{s}_{t_1} \vdash (\Lambda r | (f^{s}_{t_1} : t_1)) t_1^{t} : t_1^{t} \) \hspace{1cm} [(5.11), (5.12), E8]

5.14. \( ((f^{s}_{t_1} : t_1))^{r}_{t_1}^{t} t_1^{t} \) \hspace{1cm} [(5.3b)]

5.15. \( \vdash A_2; t^{s}_{t_1} \vdash (\Lambda r | (f^{s}_{t_1} : t_1)) t_1^{t} : t_1^{t} \) \hspace{1cm} [(5.13), (5.14), ER9]

5.16. \( \vdash A_2; t^{s}_{t_1} \vdash e^{t}_{t_1} : t_1^{t} \) \hspace{1cm} [(5.1), (5.15)]

\[\square\]

**Theorem 2.5.3.** [renaming bound type variables]

Let \( A \in \text{Assumptions}; s,t \in \text{Tvar}; t_1, t_2, t_3, \ldots \in \text{Texp} \) and \( e \in \text{Exp} \). Then the following inference rules can be derived:

\[
\text{ER15.1.} \quad \frac{\vdash A \vdash v(\Lambda t | t e) \quad \text{provided } s \in \text{FTV}(te)}{\vdash A \vdash v(\Lambda s | t e)}
\]

\[
\text{ER15.2.} \quad \frac{\vdash A \vdash (\Lambda t | e) t_1 : t_2 \quad \text{provided } s \in \text{FTV}(e \colon \tau_{A; t}(e))}{\vdash A \vdash (\Lambda s | e) t_1 : t_2}
\]
Proof.

Assume $s \in \text{FTV}(te)$

\[ \vdash A \triangleright v(A \ t \ | \ te) \]  

\[ (*) \]  

1. \[ \vdash A; t \triangleright te \]  

2. \[ \vdash A; t; s \triangleright te \]  

3. \[ \vdash A; s; t \triangleright te \]  

4. \[ \vdash A; s \triangleright s \]  

5. \[ \vdash A; s \triangleright te^t_s \]  

6. \[ \vdash A \triangleright v(A \ s \ | \ te^t_s) \]  

\[ (**) \]  

Hence rule \text{ER15.1} is derivable.

Assume $s \in \text{FTV}(e:te)$

\[ \vdash A \triangleright (\Lambda \ t \ | \ e)te1 : te2 \]  

\[ (*) \]  

\[ (**) \]  

1. \[ \vdash A \triangleright te1 \]  

2. Let $te \in Texp$ be such that

a) \[ \vdash A; t \triangleright e : te \]  

b) \[ te2 = \alpha te^t_{te1} \]  

3. \[ \vdash A; t; s \triangleright e : te \]  

4. \[ \vdash A; s; t \triangleright e : te \]  

5. \[ \vdash A; s \triangleright s \]  

6. \[ \vdash A; s \triangleright e^t_s : te^t_s \]  

7. \[ \vdash A \triangleright (\Lambda \ s \ | \ e^t_s)te1 : (te^t_s)te1 \]  

8. \[ \vdash A \triangleright (\Lambda \ s \ | \ e^t_s)te1 : te^t_{te1} \]  

9. \[ \vdash A \triangleright (\Lambda \ s \ | \ e^t_s)te1 : te2 \]  

\[ (**) \text{,ER8} \]  

\[ ((**),ER8) \]  

\[ ([*],(2a),ER11.1] \]  

\[ ([*],(1),ER11.1) \]  

\[ ([2a],ER11.1) \]  

\[ ([2a],ER11.1) \]  

\[ ([4a],(5),thm2.5.2) \]  

\[ ([4b],(6),ER8) \]  

\[ ([4b],(6),ER8) \]  

\[ ([*],(7),ER9,subst) \]  

\[ ([*],(7),ER9,subst) \]  

\[ ([2b],(8),ER9) \]  

\[ ([2b],(8),ER9) \]  

Hence rule \text{ER15.2} is derivable.

\[ \Box \]
CONTRARY

CHAPTER 2

Theorem 2.5A. [Substitution of expressions for variables in expressions]

Let $A \in \text{Assumptions}$; $te, tel \in Texp$; $x \in \text{Var}$ and $e, el \in \text{Exp}$. Then the following inference rule is derivable:

$$
\begin{align*}
& A \vdash e : te \\
& A;x:tel \vdash e : te \\
\Rightarrow & A \vdash e_x^{el} : te
\end{align*}

Proof. By induction on the structure of expression $e$. We consider only a few cases. The other cases are trivial.

Assume

1.1. Let $e \equiv x$

1.2. $\vdash A \triangleright te$

1.3. $\vdash A;x:tel \triangleright e : te$

1.4. te $\equiv_{\alpha} tel$

1.5. $\vdash A \triangleright x^{el}_x : te$

1.6. $\vdash A \triangleright x^{el}_x : te$

1.7. $\vdash A \triangleright e^{el}_x : te$

2.1. Let $e \equiv y \land y \neq x$

2.2. Let $A_1, A_2 \in \text{Assumptions}$ be such that

a) $A = A_1; y: te; A_2$

b) $\vdash A_1 \triangleright te$

c) $y \notin \text{FV}(A_2; x: tel)$

d) $\text{FTV}(te) \cap \text{FTV}(A_2; x: tel) = \emptyset$

2.3. a) $y \notin \text{FV}(A_2)$

b) $\text{FTV}(te) \cap \text{FTV}(A_2) = \emptyset$

2.4. $\vdash A \triangleright y : te$
2.5. \[ \vdash A \cdot \forall x_{e1} \cdot y^x_{e1} : \text{te} \]

2.6. \[ \vdash A \cdot \forall x_{e1} : \text{te} \]

3.1. Let \( e = (\lambda y: \text{te} \cdot f) \)

3.2. Let \( tf \in \text{Texp} \) be such that
   a) \[ \vdash A; x: \text{te} \cdot \text{te} \cdot \text{te} \cdot \text{te} \cdot \text{tf} \]
   b) \[ \vdash A; x: \text{te} \cdot y: \text{te} \cdot f : \text{tf} \]
   c) \( \text{te} \equiv \alpha \cdot \text{te} \cdot \text{te} \rightarrow \text{tf} \)

3.3. Let \( z \) be the first variable such that
   \( z \notin x \land z \notin \text{FV}(f) \land z \notin \text{FV}(e_{e1}) \)

3.4. \[ \vdash A; z: \text{te} \cdot y: \text{te} \cdot \text{te} \]

3.5. \[ \vdash A; z: \text{te} \cdot y: \text{te} \cdot z: \text{te} \cdot f : \text{tf} \]

3.6. \[ \vdash A; z: \text{te} \cdot y: \text{te} : \text{te} \cdot f : \text{tf} \]

3.7. \[ \vdash A; z: \text{te} \cdot y: \text{te} = \text{te} \cdot z : \text{te} \]

3.8. \[ \vdash A; z: \text{te} \cdot y: \text{te} = \text{te} \cdot z : \text{te} \]

3.9. \[ \vdash A; z: \text{te} \cdot y: \text{te} = \text{te} \cdot f : \text{te} \]

3.10. \[ \vdash A \cdot \text{te} \cdot \text{te} \cdot \text{tf} \]

3.11. \[ \vdash A; z: \text{te} \cdot \text{te} \cdot \text{te} \cdot \text{te} : \text{te} \]

3.12. \[ \vdash A; z: \text{te} \cdot \text{te} \cdot (f^y_{z\cdot e1}) : \text{te} \]

3.13. \[ \vdash A \cdot (\lambda z: \text{te} \cdot (f^y_{z\cdot e1}) : \text{te} \cdot \text{te} \rightarrow \text{tf} \]

3.14. \[ \vdash A \cdot (\lambda y: \text{te} \cdot f^y_{e1} : \text{te} \cdot \text{te} \rightarrow \text{tf} \]

3.15. \[ \vdash A \cdot e^x_{e1} : \text{te} \]

4.1. Let \( e = (\lambda s \cdot f)\text{tf} \)

4.2. \[ \vdash A; x: \text{te} \cdot \text{te} \cdot \text{tf} \]

4.3. Let \( tf \in \text{Texp} \) be such that
   a) \[ \vdash A; x: \text{te} \cdot s \cdot f : \text{tf} \]
   b) \[ \text{te} = \alpha \cdot \text{tf} \cdot \text{tf} \]

\[ [(2.1),(2.4)] \]

\[ [(2.1),(2.4)] \]

\[ [(2.1),(2.4)] \]

\[ [(\ast), \text{ER11.2}] \]

\[ [(3.2a), \text{ER11.2}] \]

\[ [(3.2b),(3.4),(3.3), \text{ER11.2}] \]

\[ [(3.5), \text{ER12.2}] \]

\[ [(3.2a), \text{ER2}] \]

\[ [(3.3),(3.7), \text{ER12.2}] \]

\[ [(3.6),(3.8), \text{IH}] \]

\[ [(3.2a)] \]

\[ [(\ast),(3.10),(3.3), \text{ER11.2}] \]

\[ [(3.9),(3.11), \text{IH}] \]

\[ [(3.10),(3.12), \text{ER6.1}] \]

\[ [(3.3),(3.13)] \]

\[ [(3.1),(3.2c),(3.14), \text{ER9}] \]

\[ [(\ast),(3.10),(3.3), \text{ER11.2}] \]

\[ [(\ast),(3.10),(3.3), \text{ER11.2}] \]

\[ [(\ast),(3.10),(3.3), \text{ER11.2}] \]

\[ [(\ast),(3.10),(3.3), \text{ER11.2}] \]
CHAPTER 2

4.4. Assume without loss of generality that

\[ s \notin \text{FTV}(el:te1) \]  

4.5. \[ \vdash A; s \triangleright e1 : te1 \]  

4.6. \[ \vdash A; x:te1 \triangleright f : tf \]  

4.7. \[ \vdash A \triangleright f^x_{e1} : tf \]  

4.8. \[ \vdash A \triangleright tf1 \]  

4.9. \[ \vdash A \triangleright (\Lambda s \triangleright f^x_{e1}) tf1 : tf^s \]  

4.10. \[ \vdash A \triangleright e^x_{e1} : te \]  

\[ \square \]  

Theorem 2.5.5.[renaming bound variables]

Let \( A \in \text{Assumptions} ; te1,te2 \in \text{Texp} ; x,y \in \text{Var} \) and \( e \in \text{Exp} \). Then the following inference rules can be derived:

\[ \text{ER17.1. } \frac{A \triangleright (\lambda x:te1 \mid e) : te2}{A \triangleright (\lambda y:te1 \mid e^x_{y}) : te2} \quad \text{provided } y \notin \text{FV}(e) \]

\[ \text{ER17.2. } \frac{A \triangleright (\Lambda s \triangleright x:te1 \mid e) : te2}{A \triangleright (\Lambda s \triangleright y:te1 \mid e^x_{y}) : te2} \quad \text{provided } y \notin \text{FV}(e) \]

Proof.
Assume \( y \notin \text{FV}(e) \)  

\[ (*) \]

\[ (**) \]

1. Let \( te \in \text{Texp} \) be such that

a) \[ \vdash A \triangleright te1 , te \]

b) \[ \vdash A; x:te1 \triangleright e : te \]

c) \( \text{te}2 = \alpha \text{te}1 \rightarrow \text{te} \)

2. \[ \vdash A; x:te1 \triangleright te1 \]  

3. \[ \vdash A; y:te1 \triangleright y : te1 \]  

[thm2.5.3] [(*),(4.4),ER11.1] [((*)),(4.4),ER11.1] [4.3a,ER12.3] [(4.5),(4.6),IH] [(4.7),(4.8),ER8] [(4.1),(4.3b),(4.9),ER9] [ER17.1] [ER17.2] [(**),ER6.1] [(1a),ER11.2] [(1a),ER2]
4. \( \vdash A; x: \text{tel}; y: \text{tel} \triangleright e : \text{te} \quad [(2),(1b),ER11.2] \\
5. \( \vdash A; y: \text{tel}; x: \text{tel} \triangleright e : \text{te} \quad [(4),ER12.2] \\
6. \( \vdash A; y: \text{tel} \triangleright e^x_y : \text{te} \quad [(3),(5),\text{thm2.5.4}] \\
7. \( \vdash A \triangleright (\lambda y: \text{tel} | e^x_y) : \text{tel} \rightarrow \text{te} \quad [(1a),(6),ER6.1] \\
8. \( \vdash A \triangleright (\lambda y: \text{tel} | e^x_y) : \text{te2} \quad [(1c),(7),ER9] \\

Hence rule ER17.1 is derivable. Similarly it can be shown that rule ER17.2 is derivable.

\( \square \)
CHAPTER 3

3. REDUCTION

3.1. Introduction.

In this chapter a reduction relation \( \rightarrow \) on expressions is defined that provides an operational semantics for our kernel language. We shall present this reduction relation in the form of a formal theory (cf. Hindley and Seldin [HiSe86]). Besides reduction rules that deal with expressions having function types, which are familiar from the lambda calculus, the theory contains reduction rules for expressions having sum, product or recursive types.

In order to present this theory we need the notion of a context. Suppose we take an expression and replace some of its subexpressions by the fresh symbol \( \$ \). The resulting term is called a context. Actually we think of a context as an expression with some holes in it. The symbol \( \$ \) merely enables us to give a proper syntactic definition. To that end replace in rules E1 – E8 of chapter 1 the nonterminal \( \text{Exp} \) by \( \text{C}\_\text{and}\_\text{E} \) and add the rule \( \text{C}\_\text{and}\_\text{E} ::= \$ \). Let \( \text{Exp} \) be the subset of sentences of \( \text{C}\_\text{and}\_\text{E} \) that contain zero occurrences of the symbol \( \$ \), and let \( \text{Context} \) be the subset of sentences that contain at least one occurrence of \( \$ \). Notice that substituting an expression for \( \$ \) describes the process of filling in the holes of a context.

3.2. The theory of reduction.

The theory of reduction consists of formula’s of the form \( \text{Exp} \rightarrow \text{Exp} \) and the following rules:

\[(v) \quad \text{(btm | te)} \rightarrow \text{(btm | te)}\]

\[(\delta) \quad \text{(down (up e))} \rightarrow e\]

\[(\sigma_1) \quad \text{(appl (sum f1 f2) (inl e1 | te2))} \rightarrow \text{(appl f1 e1)}\]

\[(\sigma_2) \quad \text{(appl (sum f1 f2) (inr te1 | e2))} \rightarrow \text{(appl f2 e2)}\]

\[(\sigma_3) \quad \text{(appls (sums f1 f2) (inls e1 | te2))} \rightarrow \text{(appls f1 e1)}\]

\[(\sigma_4) \quad \text{(appls (sums f1 f2) (inrs te1 | e2))} \rightarrow \text{(appls f2 e2)}\]
(σ₃) \((\text{sums } (λs \ x:te \ | \ (\text{appls } f \ (\text{inls } x \ | \ te2))))\)

\((λs \ x:te2 \ | \ (\text{appls } f \ (\text{inrs } te1 \ | \ x)))) \rightarrow f, \quad x \in \text{FV}(f)\)

(π₁) \((\text{prol } (\text{prod } e1 \ e2)) \rightarrow e1\)

(π₂) \((\text{pror } (\text{prod } e1 \ e2)) \rightarrow e2\)

(π₃) \((\text{prod } (\text{prol } e) \ (\text{pror } e)) \rightarrow e\)

(π₄) \((\text{prols } (\text{prods } e1 \ e2)) \rightarrow e1, \quad \text{provided } e2 \text{ in normal form}\)

(π₅) \((\text{prors } (\text{prods } e1 \ e2)) \rightarrow e2, \quad \text{provided } e1 \text{ in normal form}\)

(π₆) \((\text{prods } (\text{prols } e) \ (\text{prors } e)) \rightarrow e\)

(ε₁) \((\text{elim } v(λ t \ | \ te) \ | \ (\text{intro } v(λ t \ | \ te) \ | \ e)) \rightarrow e\)

(ε₂) \((\text{intro } v(λ t \ | \ te) \ | \ (\text{elim } v(λ t \ | \ te) \ | \ e)) \rightarrow e\)

(β₁) \((\text{appl } (λ x:tx \ | \ e) e1) \rightarrow e^x_{e1}\)

(β₂) \((\text{appls } (λ x:tx \ | \ e) e1) \rightarrow e^x_{e1}, \quad \text{provided } e1 \text{ in normal form}\)

(β₃) \((λ t \ | \ e)te1 \rightarrow e^t_{te1}\)

(η₁) \((λ x:tx \ | \ (\text{appls } f x)) \rightarrow f, \quad x \not\in \text{FV}(f)\)

(η₂) \((λ x:tx \ | \ (\text{appls } f x)) \rightarrow f, \quad x \not\in \text{FV}(f)\)

(ρ) \(e \rightarrow e\)

\(e1 \rightarrow e2\)

\(e2 \rightarrow e3\)

\(e1 \rightarrow e3\)

(τ) \(\begin{align*}
\frac{\text{e1} \rightarrow \text{e2}}{\text{e2} \rightarrow \text{e3}} \\
\frac{\text{e1} \rightarrow \text{e3}}{}
\end{align*}\)

(ψ) \(\begin{align*}
\text{e1} & \rightarrow \text{e2}, \\
\text{e1} & \rightarrow \text{e2}, \\
\text{e2} & \rightarrow \text{e3}
\end{align*}\)

Rule \(ψ\) expresses the substitutivity property (or compatibility property as it is called in Barendregt [Ba81]) of \(\rightarrow\). It states, however, one exception, viz. subexpressions appearing in an up-context can not be reduced. Hence \(\rightarrow\) is the reflexive, transitive and (almost) substitutive closure of the one-step reduction relation defined by rules \(\nu\) thru \(\eta\). The left-hand side of any of these rules is called a redex. An expression in which all redices, if any, appear inside an up-context is called a normal form.
CHAPTER 3

Notice that the notions redex and normal form are actually defined by mutual recursion, on account of the constraints in rules $\pi_4$, $\pi_5$ and $\beta_2$. In particular $(\text{btm} \mid \text{te})$ is not a normal form. This is proper, since it corresponds to a nonterminating computation that yields no information at all. On the other hand, any up-expression is in normal form.

Up-expressions can be used to enforce lazy evaluation. Consider the two expressions

$$(\text{appl} (\lambda x:tx \mid (\text{inl} x \mid \text{te2})) e)$$

and

$$(\text{appl} (\lambda x:tx \mid (\text{inl} (\text{down} x) \mid \text{te2})) (\text{up} e))$$

If $e \Rightarrow e_1$ then $(\text{appl} (\lambda x:tx \mid (\text{inl} x \mid \text{te2})) e) \Rightarrow (\text{inl} e_1 \mid \text{te2})$ in two distinct ways, viz. applying rule $\beta_1$ before rule $\psi$, which is called lazy evaluation or applying rule $\psi$ and then rule $\beta_1$, which is called eager evaluation. Likewise $(\text{appl} (\lambda x:tx \mid (\text{inl} (\text{down} x) \mid \text{te2})) (\text{up} e)) \Rightarrow (\text{inl} e_1 \mid \text{te2})$, but the order in which the rules are applied has to be first $\beta_1$ then $\delta$ and finally $\psi$.

One would expect that reduction does not change the type of an expression. This is indeed the case, if renaming of bound variables is ignored. Of course type expressions that differ only in the names of their bound variables have the same semantics. Hence, if we are a little more liberal and consider a type to be a class of type expressions that are equal up to $\alpha$-conversion then we can say that types are invariant under reduction.

**Theorem 3.2.1.**

Let $A \in \text{Assumptions}$ and $e_1,e_2 \in \text{Exp}$. If $e_1 \in \text{WTE}(A)$ and $e_1 \Rightarrow e_2$ Then $e_2 \in \text{WTE}(A)$ and $\tau_A(e_1) \equiv_{\alpha} \tau_A(e_2)$. 

26
Proof. With the exception of the $\beta$-rules this follows for each of the remaining rules $v$ thru $\eta$ by a straightforward calculation. Rules $\beta_1$ and $\beta_2$ preserve types on account of theorem 2.5.3. Rule $\beta_3$ preserves types on account of theorem 2.5.4.

Remark. For reductions $e_1 \rightarrow e_2$ that do not comprise rule $\beta_3$ one can prove that $\tau^A(e_1) \equiv \tau^A(e_2)$.

\[
\Box
\]
CHAPTER 4

4.SEMANTICS OF TYPE EXPRESSIONS

4.1. Introduction.

In this chapter we show how a complete partial order (c.p.o.) can be associated to every type expression. The c.p.o.'s corresponding to recursively defined types, i.e. type expressions of the form \( v(A \ {t|t} e) \), are found using the inverse limit construction. The use of this technique to solve recursive domain equations has been described by Smyth & Plotkin [SP82], Lehmann & Smyth [LS81] and others. A detailed description (for the case of the category of c.p.o.'s with embedding–projection pairs as morphisms) can be found in Bos & Hemerik [BH88]. For general aspects of category theory we refer to Herrlich & Strecker [HeStr73] or Maclane [McL71].

In this section we introduce some notations and conventions. Some elementary properties of the concepts introduced in this section are given in section 4.2. The actual semantics of type expressions is given in section 4.3. We first associate a certain functor with every type expression. The c.p.o. corresponding to a type expression is then found by applying that functor to an object, called the type environment. Finally in section 4.4. some elementary properties of the semantics of type expressions are given.

Let \( s, t \in T\text{var} \). In the sequel we shall use the following notations.

- \( C = \mathcal{CPO}_{PR} \), the category of c.p.o.'s with embedding/projection pairs as morphisms
- \( \Pi C = \prod_{t \in T\text{var}} \mathcal{CPO}_{PR} \).
- \( P_t : \Pi C \rightarrow C \), the projection functor on component \( t \).
- If \( A \in \text{obj}(\Pi C) \), then \( A_t = P_t(A) \).
- If \( f \in \text{mor}(\Pi C) \), then \( f_t = P_t(f) \).
- If \( A \in \text{obj}(\Pi C) \), \( B \in \text{obj}(C) \), then \( A[B/t] \in \text{obj}(\Pi C) \) is defined by
A[B/t]_s = \begin{cases} A_s & \text{if } s \neq t \\ B & \text{if } s = t \end{cases}

If \( f \in \text{mor}(\Pi C) \), \( g \in \text{mor}(C) \), then \( f[g/t] \in \text{mor}(\Pi C) \) is defined by

\[
f[g/t]_s = \begin{cases} f_s & \text{if } s \neq t \\ g & \text{if } s = t \end{cases}.
\]

Consider the functors \( F: \Pi C \to \Pi C \) and \( G: \Pi C \to C \). Then the functor \( F[G/t]: \Pi C \to \Pi C \) is defined by

\[
P_s \circ F[G/t] = \begin{cases} P_s \circ F & \text{if } s \neq t \\ G & \text{if } s = t \end{cases}.
\]

- \( \text{Id}: \Pi C \to \Pi C \), the identity functor.
- \( \text{id}_A: A \to A \), the identity morphism on object \( A \).
- Consider the functor \( F: \Pi C \to C \). The functor \( \text{abstr}_t F: \Pi C \to (C \to C) \) is defined in the following way:

i) For \( A \in \text{obj}(\Pi C) \) is \( \text{abstr}_t F(A) \) the object in the category \( C \to C \) (i.e. the functor \( C \to C \)) defined by

\[
\text{abstr}_t F(A)(B) = F(A[B/t]) \quad \text{for } B \in \text{obj}(C),
\]

\[
\text{abstr}_t F(A)(g) = F(\text{id}_A[g/t]) \quad \text{for } g \in \text{mor}(C).
\]

ii) For \( f \in \text{mor}(\Pi C) \) is \( \text{abstr}_t F(f) \) the morphism in the category \( C \to C \) (i.e. the natural transformation) defined by

\[
(\text{abstr}_t F(f))_B = F(f[\text{id}_B/t]) \quad \text{for } B \in \text{obj}(C).
\]

Suppose \( D \) is an arbitrary category. A functor \( F: \Pi C \to D \) will be called independent of \( t \) if

\[
F = F \circ \text{Id}[G/t] \quad \text{for all functors } G: \Pi C \to C.
\]

We shall use the following functors.
CHAPTER 4

\[ \text{CONST}_A : \Pi C \to C , \] the constant functor corresponding to an object \( A \in \text{obj}(C) \),

\[ \text{LIFT} : C \to C , \] the lifting functor,

\[ \text{DS} : C \times C \to C , \] the disjoint sum functor,

\[ \text{CP} : C \times C \to C , \] the cartesian product functor,

\[ \text{FS} : C \times C \to C , \] the function space functor,

\[ \text{CS} : C \times C \to C , \] the coalesced sum functor,

\[ \text{SP} : C \times C \to C , \] the smash product functor,

\[ \text{SF} : C \times C \to C , \] the strict function space functor,

\[ \text{IFP} : [C \to C] \to C , \] the initial fixed point functor.

The formal definition of these functors can be found in Bos & Hemerik [BH88] or Smyth and Plotkin [SP82].

4.2. Elementary properties.

The following properties of the concepts introduced in the preceding section can easily be shown. Let \( F,G : \Pi C \to C , H : C \to D \) and \( t,u \in \text{Tvar} \). Then

- \( F = P_t \circ \text{Id}[F/t] \), \hspace{1cm} (4.2.1)
- \( \text{If } t \neq u \text{ then } P_u \text{ is independent of } t \), \hspace{1cm} (4.2.2)
- \( \text{abstr}_u F \text{ is independent of } u \), \hspace{1cm} (4.2.3)
- \( \text{If } F \text{ is independent of } t \text{ then } \text{abstr}_u F \text{ is independent of } t \), \hspace{1cm} (4.2.4)
- \( \text{If } F \text{ is independent of } u \text{ then } \text{abstr}_u (F \circ \text{Id}[P_u/t]) = \text{abstr}_t F \), \hspace{1cm} (4.2.5)
- \( \text{If } G \text{ is independent of } u \text{ and } t \neq u \text{, then } \text{abstr}_u (F \circ \text{Id}[G/t]) = (\text{abstr}_u F) \circ \text{Id}[G/t]. \) \hspace{1cm} (4.2.6)
4.3. Definition of semantics of type expression.

We first show that with every type expression an $\omega$-continuous functor $\Pi C \to C$ can be associated. Define $F : \text{Typ} \to [\Pi C \to C]$ by

- $F[\Omega] = \text{CONST}_A$, where $A$ is the one-point c.p.o.
- $F[t] = P_t$,
- $F[t1 + t2] = DS \circ F[t1] \cup F[t2]$,
- $F[t1 \times t2] = CP \circ F[t1] \times F[t2]$,
- $F[t1 \rightarrow t2] = FS \circ F[t1] \rightarrow F[t2]$,
- $F[t1 \otimes t2] = CS \circ F[t1] \otimes F[t2]$,
- $F[t1 \bigcirc t2] = SP \circ F[t1] \bigcirc F[t2]$,
- $F[v(A t|te)] = IFP \circ (\text{abstr}_t F[te])$.

The constant and projection functors are trivially $\omega$-continuous. The $\omega$-continuity of the functors $DS$, $CP$, $FS$, $CS$, $SP$ and $SF$ follows from the local continuity of the corresponding functors on $\text{CPO} \times \text{CPO}$ respectively $\text{CPO} \downarrow \times \text{CPO} \downarrow$, see for instance Smyth & Plotkin [SP82] or Bos & Hemerik [BH88]. The $\omega$-continuity of the functor $\text{LIFT}$ follows from the local continuity of the corresponding functor $\text{CPO} \to \text{CPO} \downarrow$, see also [SP82] or [BH88]. Further if $F : [\Pi C \to C]$, then also $\text{abstr}_t F : [\Pi C \to [C \to C]]$, see for instance Herrlich & Strecker [HeStr73, th.15.9]. The $\omega$-continuity of the initial fixed point functor $\text{IFP}$ is shown in Lehmann & Smyth [LS81]. Now using the property that the composition of two $\omega$-continuous functors is again $\omega$-continuous (see Mac Lane [McL71]), it is easily shown by induction on the structure of $te$ that $F[te]$ is an $\omega$-continuous functor for every type expression $te$.

Define $\text{Tenv} = \text{obj}(\Pi C)$. Elements of $\text{Tenv}$ will be called type environments. If $\rho \in \text{Tenv}$, then $\rho_t = P_t(\rho)$ is the c.p.o. associated to $t \in \text{Tvar}$ by the type environment $\rho$. The c.p.o.
CHAPTER 4

corresponding to a type expression \( \text{te} \) in the environment \( \rho \) is given by \( \mathcal{F}[\text{te}]\rho \).

4.4. Properties of the type semantics.

We now describe some properties of the semantics of type expressions. Theorem 4.4.4. shows that the functor associated to a type expression \( \text{te} \) depends only on the type variables which appear freely in \( \text{te} \). Hence the c.p.o. which corresponds to \( \text{te} \) in an environment \( \rho \) depends only on the values of \( \rho \) on \( \text{FTV}(\text{te}) \).

Theorem 4.4.1.

Let \( \text{te} \in \text{Texp} \) and \( t \in \text{Tvar} \). If \( t \in \text{FTV}(\text{te}) \) then \( \mathcal{F}[\text{te}] \) is independent of \( t \).

Proof. The theorem is easily proved using induction on the structure of \( \text{te} \).

i) \( \text{te} = \Omega \), then \( \mathcal{F}[\text{te}] = \text{CONST}_A \), where \( A \) is the one-point c.p.o. Clearly this functor is independent of \( t \).

ii) \( \text{te} = u \in \text{Tvar} \) with \( u \neq t \). Then \( \mathcal{F}[\text{te}] = P_u \), which by property (4.4.2) is independent of \( t \).

iii) \( \text{te} = \text{te}_1 + \text{te}_2 \), \( \text{te} = \text{te}_1 \times \text{te}_2 \), \( \text{te} = \text{te}_1 \rightarrow \text{te}_2 \), \( \text{te} = \text{te}_1 \odot \text{te}_2 \) and \( \text{te} = \text{te}_1 \ominus \text{te}_2 \). These cases are easily handled using the induction hypothesis that \( \mathcal{F}[\text{te}_1] \) respectively \( \mathcal{F}[\text{te}_1] \) and \( \mathcal{F}[\text{te}_2] \) are independent of \( t \).

iv) \( \text{te} = v(\Lambda \ u \mid \text{te}) \). Then \( \mathcal{F}[v(\Lambda \ u \mid \text{te})] = \text{IFP} \circ (\text{abstr}_u \mathcal{F}[\text{te}]) \). If \( t = u \) the result follows from property (4.2.3). If \( t \neq u \) then \( t \in \text{FTV}(\text{te}_1) \) and the theorem follows from the induction assumption and property (4.2.4).

\( \Box \)

The next theorem gives the behaviour of \( \mathcal{F}[\text{te}] \) under substitution in \( \text{te} \).

32
Theorem 4.4.2. [substitution in type expressions]

Let $t_1, t_2 \in \text{Texp}$ and $t \in \text{Tvar}$. Then $\mathcal{F}[t_1 \downarrow_{t_2}] = \mathcal{F}[t_1] \circ \text{Id}[\mathcal{F}[t_2] / t]$. 

**Proof.** The proof is done by induction on the structure of $t_1$.

i) $t_1 = \Omega$ or $t_1 = s$ with $s \in \text{Tvar}$ and $s \not\in t$. In these cases $t \in \text{FTV}(t_1)$ and the theorem follows from theorem 4.4.1.

ii) $t_1 = t$. A simple calculation yields that

\[
\mathcal{F}[t \downarrow_{t_2}] = P_t \circ \text{Id}[\mathcal{F}[t_2] / t] \quad \text{[property 4.2.1]}
\]

\[
= \mathcal{F}[t] \circ \text{Id}[\mathcal{F}[t_2] / t].
\]

iii) $t_1 = \uparrow t$. Then we have

\[
\mathcal{F}[(\uparrow t) \downarrow_{t_2}] = \mathcal{F}[(\uparrow t) \downarrow_{t_2}] = \text{LIFT} \circ \mathcal{F}[t \downarrow_{t_2}]
\]

\[
= \text{LIFT} \circ \mathcal{F}[t] \circ \text{Id}[\mathcal{F}[t_2] / t] \quad \text{[induction hypothesis]}
\]

\[
= \mathcal{F}[\uparrow t] \circ \text{Id}[\mathcal{F}[t_2] / t].
\]

iv) $t_1 = te_3 \& te_4$ where $\& = +, x, \rightarrow, \& \&, \text{corresponds to respectively } FU = DS, CP, FS, CS, SP, SF$. The result follows from the following computation.

\[
\mathcal{F}[(te_3 \& te_4) \downarrow_{te_2}] = \mathcal{F}[te_3 \downarrow_{te_2} \& te_4 \downarrow_{te_2}]
\]

\[
= FU \circ <\mathcal{F}[te_3] \circ \text{Id}[\mathcal{F}[te_2] / t], \mathcal{F}[te_4] \circ \text{Id}[\mathcal{F}[te_2] / t]> \quad \text{[induction hypothesis]}
\]

\[
= FU \circ <\mathcal{F}[te_3], \mathcal{F}[te_4] > \circ \text{Id}[\mathcal{F}[te_2] / t] \quad \text{[<F1\circ F, F2\circ F> = <F1,F2>\circ F]}
\]

\[
= \mathcal{F}[te_3 \& te_4] \circ \text{Id}[\mathcal{F}[te_2] / t].
\]

v) $t_1 = \nu(A s | te)$. Let $u$ be the first variable such that $u \not\in t$ and $u \not\in \text{FTV}(te) \cup \text{FTV}(te_2)$. The result now follows from the following calculation.
CHAPTER 4

\[ \mathcal{F}[ (\forall s \mid t e)_u^{t e_2} ] \]

\[ = \mathcal{F}[ (\forall u \mid (t e_2^{s})_u^{t e_2} ] \quad \text{[def. of substitution]} \]

\[ = \text{IFP} \circ (\text{abstr}_u \mathcal{F}[ (t e_2^{s})_u^{t e_2} ] ) \]

\[ = \text{IFP} \circ (\text{abstr}_u \mathcal{F}[ t e_2^{s} ] \circ \text{Id}[\mathcal{F}[ t e_2 ] / t ] ) \quad \text{[induction hypothesis]} \]

\[ = \text{IFP} \circ (\text{abstr}_u \mathcal{F}[ t e_2 ] ) \circ \text{Id}[\mathcal{F}[ t e_2 ] / t ] \quad [\mathcal{F}[ t e_2 ] \text{ is independent of } u \text{, property (4.2.6)}] \]

\[ = \text{IFP} \circ (\text{abstr}_u \mathcal{F}[ t e ] ) \circ \text{Id}[\mathcal{F}[ t e_2 ] / t ] \quad [\mathcal{F}[ t e ] \text{ is independent of } u \text{, property (4.2.5)}] \]

\[ = \mathcal{F}[ (\forall s \mid t e) ] \circ \text{Id}[\mathcal{F}[ t e_2 ] / t ] . \]

\[ \square \]

As a consequence of theorem 4.4.2 we have

\[ \mathcal{F}[ t e_1^{t} ] \rho = \mathcal{F}[ t e_1 ] (\rho[\mathcal{F}[ t e_2 ] \rho / t ] ) \quad (4.4.3) \]

for all \( t e_1, t e_2 \in \text{Texp} \), \( t \in \text{Tvar} \) and \( \rho \in \text{Tenv} \). This relation shows that substitution in a type expressions can be replaced by substitution in the type environment.

As expected, the semantics of a recursively defined type does not depend on the name of the bound variable.

Theorem 4.4.4.

Let \( t e \in \text{Texp} \) and \( t, u \in \text{Tvar} \). If \( u \notin \text{FTV}(t e) \), then

\[ \mathcal{F}[ (\forall t \mid t e) ] = \mathcal{F}[ (\forall u \mid t e_2^{t} ) ] . \]

Proof. Using the previous theorem this result can be proved by a straightforward calculation.

\[ \mathcal{F}[ (\forall t \mid t e_2^{t} ) ] \]

\[ = \text{IFP} \circ \text{abstr}_u \mathcal{F}[ t e_2^{t} ] \]

\[ = \text{IFP} \circ \text{abstr}_u ( \mathcal{F}[ t e ] ) \circ \text{Id}[P_u / t ] ) \quad \text{[theorem 4.4.2.]} \]
Finally we mention a technical result which will be used in section 5. From part v) of the proof of theorem 4.4.2. we infer that if \( u \not= t \) and \( t \not\in \text{FTV}(te) \cup \text{FTV}(te2) \), then
\[
\text{abstr}_u \mathcal{F}[\langle te_u \rangle_{te2}] = (\text{abstr}_s \mathcal{F}[te]) \circ \text{Id}[\mathcal{F}[te2] / t].
\]
Hence we see that under the same assumptions
\[
(\text{abstr}_u \mathcal{F}[\langle te_u \rangle_{te2}] \rho = \text{abstr}_s \mathcal{F}[te]) \circ (\rho[\mathcal{F}[te2] \rho / t]) \tag{4.4.5}
\]
for every type assignment \( \rho \).
CHAPTER 5

5. SEMANTICS OF EXPRESSIONS

5.1. States.

The value of an expression \( e \in WTE(A) \) depends on the values of the free variables occurring in it. The function that defines these values is called a state. Hence a state maps each free variable of an expression to an element of a specific c.p.o. Which c.p.o. that is depends on the assumption \( A \) and the type environment \( \rho \). Therefore we define for \( A \in \text{Assumptions} \) and \( \rho \in \text{Tenv} \)

\[
ST_{\rho,A} = \Pi \{ \mathcal{F}[\tau_A(x)]\rho \mid x \in WTV(A) \} \tag{5.1.1}
\]

i.e. the set of functions \( \sigma \) such that \( \sigma(x) \in \mathcal{F}[\tau_A(x)]\rho \) for all \( x \in WTV(A) \). Elements of \( ST_{\rho,A} \) are called states.

Definition 5.1.2.

Let \( A \in \text{Assumptions} \) and \( \rho \in \text{Tenv} \). Moreover, let \( x \in \text{Var} \) and \( tx \in \text{Texp} \) such that \( \vdash A \triangleright tx \) and let \( d \in \mathcal{F}[tx]\rho \). Then for \( \sigma \in ST_{\rho,A} \) we define the function \( \sigma[d/x] \in ST_{\rho,A;x:tx} \) by:

\[
\sigma[d/x](y) = \begin{cases} 
\text{if } y = x \rightarrow d & \text{if } y \neq x \rightarrow \sigma(y)
\end{cases}
\]

Moreover, for \( A_1 \in \text{Assumptions} \) and \( \rho_1 \in \text{Tenv} \) such that \( WTV(A_1) \leq WTV(A) \) and \( \mathcal{F}[\tau_{A_1}(x)]\rho_1 = \mathcal{F}[\tau_A(x)]\rho \) for all \( x \in WTV(A_1) \) we define the restriction \( \sigma \uparrow WTV(A_1) \in ST_{\rho_1,A_1} \) by :

\[
(\sigma \uparrow WTV(A_1))(x) = \sigma(x)
\]

Note that if also \( \vdash A_1 \triangleright tx \) then

\[
\sigma[d/x] \uparrow (WTV(A_1;x:tx)) = (\sigma \uparrow WTV(A_1))[d/x] \tag{5.1.3}
\]
5.2. Semantic mappings

The meaning of an expression $e$ is given by a family of mappings $\mathcal{E} = \langle \mathcal{E}_{\rho,A} \mid \rho \in \text{Tenv}, A \in \text{Assumptions} \rangle$ such that for $\rho$ and $A$ the domain of $\mathcal{E}_{\rho,A}$ is $\text{WTE}(A)$ and for all expressions $e \in \text{WTE}(A)$ we have $\mathcal{E}_{\rho,A}[e] \in \text{ST}_{\rho,A} \rightarrow \mathcal{F}[\tau_A(e)] \rho$. Hence given a state $\sigma \in \text{ST}_{\rho,A}$, $\mathcal{E}_{\rho,A}[e] \sigma$ indeed yields a value in the domain $\mathcal{F}[\tau_A(e)] \rho$.

Definition 5.2.1. [Semantic mapping $\mathcal{E}_{\rho,A}$]

Let $\rho \in \text{Tenv}$ and $A \in \text{Assumptions}$. For all $\tau, \tau_1 \in \text{Var}; \tau, \tau_1 \in \text{Exp}; x \in \text{Var}; e, e_1, e_2, f_1, f_2 \in \text{Exp}$ and $\sigma \in \text{ST}_{\rho,A}$ the mapping $\mathcal{E}_{\rho,A} \in \Pi \{ \text{ST} \rho,A \rightarrow \mathcal{F}[\tau_A(e)] \rho \mid e \in \text{WTE}(A) \}$ is defined by:

1. $\mathcal{E}_{\rho,A}[(\text{btm} | \tau_1)]\sigma = \uparrow_D$
   where $D = \mathcal{F}[\tau_1] \rho$

2. $\mathcal{E}_{\rho,A}[x] \sigma = \sigma(x)$

3.1. $\mathcal{E}_{\rho,A}[(\text{up } e)] \sigma = <0, \mathcal{E}_{\rho,A}[e] \sigma> \uparrow_D$
   where $D = \mathcal{F}[\tau_A(x)] \rho$

3.2. $\mathcal{E}_{\rho,A}[(\text{down } e)] \sigma =$
   if $\mathcal{E}_{\rho,A}[e] \sigma = \uparrow_D \rightarrow \uparrow_D$
   \[ \mathcal{E}_{\rho,A}[e] \sigma = <0, d> \uparrow_D \rightarrow d \]
   \fi
   where $D = \mathcal{F}[\tau_A(x)] \rho$

4.1. $\mathcal{E}_{\rho,A}[(\text{inl } e_1 | \tau_2)] \sigma = <1, \mathcal{E}_{\rho,A}[e_1] \sigma>_{D_1 + D_2}$

\[ \mathcal{E}_{\rho,A}[(\text{inls } e_1 | \tau_2)] \sigma = <1, \mathcal{E}_{\rho,A}[e_1] \sigma>_{D_1 \Theta D_2} \]
   where $D_1 = \mathcal{F}[\tau_A(e_1)] \rho$, $D_2 = \mathcal{F}[\tau_2] \rho$
4.2. $\mathcal{E}_{\rho,A}[\text{inr } t_1 \mid e_2] \sigma = <2, \mathcal{E}_{\rho,A}[e_2] \sigma >_{D_1+D_2}$

$\mathcal{E}_{\rho,A}[\text{inrs } t_1 \mid e_2] \sigma = <2, \mathcal{E}_{\rho,A}[e_2] \sigma >_{D_1 \otimes D_2}$

where $D_1 = \mathcal{F}[t_1] \rho$, $D_2 = \mathcal{F}[\tau_A(e_2)] \rho$

note that $<1,1\uparrow D_1 \otimes D_2 > = <1\uparrow D_1 \otimes D_2 >_{D_1 \otimes D_2}$.

4.3. $\mathcal{E}_{\rho,A}[\text{sum } f_1 f_2] \sigma =$

$(\lambda d \in D_1+D_2$

$\mid$ if $d = <1, d_1 >_{D_1+D_2}$ $\rightarrow <1 \downarrow D$

$\mid$ $d = <1, d_1 >_{D_1+D_2}$ $\rightarrow (\mathcal{E}_{\rho,A}[f_1] \sigma)(d_1)$

$\mid$ $d = <2, d_2 >_{D_1+D_2}$ $\rightarrow (\mathcal{E}_{\rho,A}[f_2] \sigma)(d_2)$

$\mid$ fi)

where $D_1 \rightarrow D = \mathcal{F}[\tau_A(f_1)] \rho$, $D_2 \rightarrow D = \mathcal{F}[\tau_A(f_2)] \rho$

$\mathcal{E}_{\rho,A}[\text{sums } f_1 f_2] \sigma =$

$(\lambda d \in D_1 \otimes D_2$

$\mid$ if $d = <1, d_1 >_{D_1 \otimes D_2}$ $\rightarrow (\mathcal{E}_{\rho,A}[f_1] \sigma)(d_1)$

$\mid$ $d = <2, d_2 >_{D_1 \otimes D_2}$ $\rightarrow (\mathcal{E}_{\rho,A}[f_2] \sigma)(d_2)$

$\mid$ fi)

where $D_1 \rightarrow D = \mathcal{F}[\tau_A(f_1)] \rho$, $D_2 \rightarrow D = \mathcal{F}[\tau_A(f_2)] \rho$

5.1. $\mathcal{E}_{\rho,A}[\text{prol } e] \sigma = \pi_1(\mathcal{E}_{\rho,A}[e] \sigma)$

$\mathcal{E}_{\rho,A}[\text{proor } e] \sigma = \pi_2(\mathcal{E}_{\rho,A}[e] \sigma)$

where $\pi_1 = (\lambda <d_1,d_2 >_{D_1 \times D_2} \in D_1 \times D_2 \mid d_1)$

and $\pi_2 = (\lambda <d_1,d_2 >_{D_1 \times D_2} \in D_1 \times D_2 \mid d_2)$

and $D_1 \times D_2 = \mathcal{F}[\tau_A(e)] \rho$
5.2. \( \mathcal{E}_{\rho,A}[(\text{pros } e)] \sigma = \psi_1(\mathcal{E}_{\rho,A}[e] \sigma) \)

\[ \begin{align*}
\mathcal{E}_{\rho,A}[(\text{pros } e)] \sigma &= \psi_2(\mathcal{E}_{\rho,A}[e] \sigma) \\
\text{ where } \psi_1 &= (\lambda <d_1,d_2> \in D_1 \otimes D_2 | \text{ if } d_2 = 1 \text{ then } D_1 \not\supseteq D_2, d_1 \not\supseteq d_2) \text{ and } d_1 \not\supseteq D_1 \not\supseteq d_2 \\
\text{ and } \psi_2 &= (\lambda <d_1,d_2> \in D_1 \otimes D_2 | \text{ if } d_1 = 1 \text{ then } D_2 \not\supseteq D_1 \not\supseteq d_2) \text{ and } D_1 \not\supseteq D_2 = \mathcal{F}[\tau_A(e)] \rho
\end{align*} \]

Note that \( <d_1,1,D_2> \subseteq D_1 \otimes D_2 = 1 \otimes D_2 = <1,D_1,d_2>^2D_1 \otimes D_2 \)

5.3. \( \mathcal{E}_{\rho,A}[(\text{prod } e1 \text{ e2})] \sigma = <\mathcal{E}_{\rho,A}[e1] \sigma, \mathcal{E}_{\rho,A}[e2] \sigma> \mathcal{D}_1 \mathcal{X} D_2 \)

\( \mathcal{E}_{\rho,A}[(\text{prods } e1 \text{ e2})] \sigma = <\mathcal{E}_{\rho,A}[e1] \sigma, \mathcal{E}_{\rho,A}[e2] \sigma> D_1 \otimes D_2 \)

6.1. \( \mathcal{E}_{\rho,A}[(\lambda \text{ x:tx } | \text{ e})] \sigma = (\lambda d \in D | \mathcal{E}_{\rho,A}[e] \sigma(d/x)) \)

\( \mathcal{E}_{\rho,A}[(\lambda s \text{ x:tx } | \text{ e})] \sigma = \\
(\lambda d \in D \\
\text{ if } d = 1 \longrightarrow 1 \text{ E} \\
\text{ if } d \not\supseteq 1 \longrightarrow \mathcal{E}_{\rho,A}[e] \sigma(d/x) \\
\text{ fi}) \\
\text{ where } A_1 = A;x:tx , D = \mathcal{F}[tx] \rho , E = \mathcal{F}[\tau_{A_1}(e)] \rho \)

6.2. \( \mathcal{E}_{\rho,A}[(\text{appl f e})] \sigma = \mathcal{E}_{\rho,A}[f] \sigma (\mathcal{E}_{\rho,A}[e] \sigma) \)

6.3. \( \mathcal{E}_{\rho,A}[(\text{appls f e})] \sigma = \mathcal{E}_{\rho,A}[f] \sigma (\mathcal{E}_{\rho,A}[e] \sigma) \)

7. \( \mathcal{E}_{\rho,A}[(\text{intro } v(\Lambda t | te) | e)] \sigma = \alpha^R(\mathcal{E}_{\rho,A}[e] \sigma) \)

\( \mathcal{E}_{\rho,A}[(\text{elim } v(\Lambda t | te) | e)] \sigma = \alpha^L(\mathcal{E}_{\rho,A}[e] \sigma) \)

where \((D,(\alpha^L,\alpha^R))\) is the initial fixed point of the endofunctor \( F = (\text{abstr}_t \mathcal{F}[\text{te}]) \rho \) on the category \( C = \mathbf{CPO}_{\mathbf{PR}} \) obtained by applying the inverse limit construction to the \( \omega \text{- chain } <F^n \downarrow C, F^n u | 0 \leq n > \) with \( u \) the unique morphism from \( \downarrow C \) to \( F(\downarrow C) \). Note that \( D = \mathcal{F}[v(\Lambda t | te)] \rho , F(D) = \mathcal{F}[te^t_v(\Lambda t | te)] \rho , \alpha^L \in \text{Hom}(F(D),D) \)
CHAPTER 5

and $\alpha^R \in \text{Hom}(D,F(D))$, cf[BH88,SP82]

8. $\mathcal{E}_{\rho_1A_1}(\sum_{1 \leq i \leq 1} e_i) = \mathcal{E}_{\rho_1A_1}[e_i](\sigma \upharpoonright \text{WTE}(A_1))$

where $\rho_1 = \rho[\mathcal{F}[te_1] / t]$, $A_1 = A; t$

Remark. All clauses of definition 2.5.2 are of the form

$\mathcal{E}_{\rho_1A_1}[e_i] = \phi(\mathcal{E}_{\rho_1A_1}[e_i], \ldots, \mathcal{E}_{\rho_nA_n}[e_n], 0 \leq n$

where $e_1, \ldots, e_n$ are the constituting subexpressions of $e$, and $\phi$ is some function. This is a proper definition iff

- if $e \in \text{WTE}(A)$ then $e_i \in \text{WTE}(A_i)$, for $1 \leq i \leq n$
- $\phi: \mathcal{F}[\tau_{A_1}(e_1)]\rho_1 \times \ldots \times \mathcal{F}[\tau_{A_n}(e_n)]\rho_n \rightarrow \mathcal{F}[\tau_A(e)]\rho$

For all clauses but 7 and 8 this is trivial. For clause 7 we consider the case $(\text{elim } v(A t | te) | e)$ only. The case $(\text{intro } v(A t | te) | e)$ will then be evident. For all $A \in \text{Assumptions}$ such that $(\text{elim } v(A t | te) | e) \in \text{WTE}(A)$:

(i) $\mathcal{F}[\tau_A((\text{elim } v(A t | te) | e))]\rho$

$= \mathcal{F}[v(A t | te)]\rho$

$= (\text{IFP } \circ (\text{abstr}_t \mathcal{F}[te]))\rho$

$= \text{IFP}((\text{abstr}_t \mathcal{F}[te])\rho)$

$= \text{IFP}(F)$

By rule ER7.2 it follows that $e \in \text{WTE}(A)$ and, moreover,

(ii) $\mathcal{F}[\tau_A(e)]\rho$

$= \mathcal{F}[te_{v(A t | te)}]\rho$

$= \mathcal{F}[te]\rho[\mathcal{F}[v(A t | te)]\rho / t]$

$= \mathcal{F}[te]\rho[\text{IFP}(F) / t]$

$= ((\text{abstr}_t \mathcal{F}[te])\rho) \text{IFP}(F)$

$= F(\text{IFP}(F))$

40
Since $\alpha^L$ is an embedding from $F(IFP(F))$ into $IFP(F)$ it follows that clause 7 is a proper
definition. From rule ER8 it follows that if $(A \vdash e)te1 \in WTE(A)$ then $e \in WTE(A_1)$. Since the
introduction of the rightmost type variable $t$ in $A_1$ invalidates type assignments for variables in
which the type expression depends on $t$ and that occur to the left of it (see rule ER2), it follows that
$WTV(A_1) \subseteq WTV(A)$. Moreover, for $x \in WTV(A_1)$ it holds that $\tau_{A_1}(x) = \tau_A(x)$ and that
$t \not\in FTV(\tau_{A_1}(x))$. Hence
\[
\begin{align*}
\mathcal{H}[\tau_{A_1}(x)]\rho_1 \\
= \mathcal{H}[\tau_{A_1}(x)]\rho[\mathcal{H}[te1]\rho / t] \\
= \mathcal{H}[\tau_{A_1}(x)]\rho \\
= \mathcal{H}[\tau_A(x)]\rho
\end{align*}
\]
and therefore $\sigma \upharpoonright WTV(A_1) \in \mathcal{ST}_{\rho_1,A_1}$ is properly defined.
\[\square\]

In the sequel we shall frequently need to compare the meanings (values) of a single
expression under similar assumptions and in similar states. The following property indicates that if
these similarities are strong enough the respective values are equal.

Property 5.2.2.

For all $A_1,A_2 \in \text{Assumptions}; \rho \in \mathcal{Tenv}; e \in \mathcal{Exp}; \sigma_1 \in \mathcal{ST}_{\rho,A_1}$ and $\sigma_2 \in \mathcal{ST}_{\rho,A_2}$:

If

$\vdash A_1 \triangleright e : te$

$\vdash A_2 \triangleright e : te$

$\sigma_1 \upharpoonright WTV(A_2) = \sigma_2 \upharpoonright WTV(A_1)$

Then

$\mathcal{E}_{\rho,A_1}[e]\sigma_1 = \mathcal{E}_{\rho,A_2}[e]\sigma_2$

\[\square\]
CHAPTER 5

5.3. Substitution and α-conversion.

In order to prove the soundness of the β-reduction rules (see chapter 6) we have to determine the meaning of expressions containing substitutions. For each of the two kinds of substitutions in expressions (see chapter 1) we present a substitution theorem.

Theorem 5.3.1. [Modification of type environment]

For all \( A \in \text{Assumptions} \); \( \rho \in \text{Tenv} \); \( t \in \text{Tvar} \); \( te \in \text{Texp} \); \( e \in \text{Exp} \); \( D \in \text{Obj}(\text{CPO}_{\text{PR}}) \) and \( \sigma \in \text{ST}_{\rho,A} \):

If \( \vdash A \triangleright e:te \)

\( t \in \text{FTV}(e:te) \) \hspace{2cm} (*)

Then \( \mathcal{E}_{\rho,A}[e]_{\sigma} = \mathcal{E}_{\rho_1,A_1}[e]_{\sigma_1} \)

where \( \rho_1 = \rho[D/t] \), \( A_1 = A;t \) and \( \sigma_1 = \sigma \vdash \text{WTV}(A_1) \)

Proof. By induction on the structure of expression \( e \). We prove only a limited number of difficult cases. Assume (*) and (**).

1.1. Let \( e \equiv x \)

1.2. \( x \in \text{WTV}(A_1) \) \hspace{2cm} [(*),(**),ER11.1]

1.3. \( \mathcal{E}_{\rho,A}[e]_{\sigma} = \sigma(x) = \sigma_1(x) = \mathcal{E}_{\rho_1,A_1}[x]_{\sigma_1} \) \hspace{2cm} [(1.2), def.\( \mathcal{E} \)]

2.1. Let \( e \equiv (\lambda y : ty \mid f) \)

2.2. Let \( d \in \mathcal{H}[ty]_{\rho} \). Moreover, let \( A_2 \in \text{Assumptions} \) and \( \sigma_2 \in \text{ST}_{\rho,A_2} \) be such that

\[ A_2 = A;y:ty \quad \sigma_2 = \sigma[d/y] \]

2.3. Let \( tf \in \text{Texp} \) be such that \hspace{2cm} [(*),(2.2),ER6.1]

\( a) \vdash A \vdash ty \), \( tf \)

\( b) \vdash A_2 \vdash f : tf \)

\( c) te \equiv_{\alpha} ty \rightarrow tf \)

2.4. \( \text{FTV}(e:te) = \text{FTV}(ty) \cup \text{FTV}(f) \cup \text{FTV}(tf) \) \hspace{2cm} [(2.1),(2.3c)]

42
2.5. \( a) t \in \text{FTV}(\text{ty}) \) 
\( b) t \in \text{FTV}(f:tf) \) 

2.6. \( \vdash A_2; t \triangleright f : tf \) 

2.7. \( \vdash A_1; ty \triangleright f : tf \) 

2.8. \( \text{WTV}(A_2; t) = \text{WTV}(A_1; ty) \) 

2.9. \( (\sigma_2 \uparrow \text{WTV}(A_2; t)) \uparrow \text{WTV}(A_1; ty) \) 
\( = (\sigma[d/y] \uparrow \text{WTV}(A_2; t)) \uparrow \text{WTV}(A_1; ty) \) 
\( = (\sigma[d/y] \uparrow (\text{WTV}(A_1; ty)) \uparrow \text{WTV}(A_2; t)) \) 
\( = (\sigma \uparrow \text{WTV}(A_1)) [d/y] \uparrow \text{WTV}(A_2; t) \) 
\( = \sigma_1[d/y] \uparrow \text{WTV}(A_2; t) \) 

2.10. 
\[ \mathcal{E}_{p,A}[(\lambda \ y:ty \mid f)] \sigma \] 
\( = (\lambda d \in \mathcal{F}[ty] \rho \mid \mathcal{E}_{p,A_2}[f] \sigma_2) \) 
\( = (\lambda d \in \mathcal{F}[ty] \rho \mid \mathcal{E}_{p,A_2}[f] \sigma_2 \uparrow \text{WTV}(A_2; t)) \) 
\( = (\lambda d \in \mathcal{F}[ty] \rho \mid \mathcal{E}_{p,A_1}[f] \sigma_1[d/y]) \) 
\( = \mathcal{E}_{p,A_1}[(\lambda y:ty \mid f)] \sigma_1 \) 

3.1. Let \( e \equiv (\text{elim} v(\Lambda s | tf) \mid f) \) 

3.2. \( a) \vdash A \triangleright f : tf \) 
\( \text{v}(\Lambda s | tf) \) 
\( b) t e = \alpha \ \text{v}(\Lambda s | tf) \) 

3.3. 
\( \text{FTV}(f:tf) \) 
\( = \text{FTV}(f) \cup (\text{FTV}(f) \setminus \{s\}) \cup \text{FTV}(\text{v}(\Lambda s | tf)) \) 
\( = \text{FTV}(f) \cup \text{FTV}(\text{v}(\Lambda s | tf)) \) 
\( = \text{FTV}((\text{elim} v(\Lambda s | tf) \mid f) : \text{v}(\Lambda s | tf)) \) 
\( = \text{FTV}(e:te) \) 

3.4. \( t \in \text{FTV}(f:tf) \land t \in \text{v}(\Lambda s | tf) \) 

3.5. Let \( (\Lambda_s, (\alpha^L_s, \alpha^R_s)) \) be the unique IFP resulting from the inverse limit construction with functor \( \text{abstr}_s \mathcal{F}[(te)] \rho \)
CHAPTER 5

3.6. Let \((B, (\beta^L, \beta^R))\) be the unique IFP resulting from the inverse limit construction with functor \((\text{abstr}_s \mathcal{I))(\text{te})\rho[D/t]\)

3.7. Since \(t \in \text{FTV}(v(A s | tf))\), by (3.4), it follows that the functor \((\text{abstr}_s \mathcal{I}(tf))\rho \) is independent of \(t\), i.e. \((\text{abstr}_s \mathcal{I}(tf))\rho = (\text{abstr}_s \mathcal{I}(tf))\rho[D/t]\)

3.8. \(\alpha^L = \beta^L\) \[\text{[(3.5),(3.6),(3.7)]}\]

3.9. 
\[
\mathcal{E}_{\rho, A}[(\text{elim } v(A s | tf) \mid f)] \sigma \\
= \alpha^L (\mathcal{E}_{\rho, A}[f]) \sigma \quad \text{[def.}\mathcal{E}\text{]} \\
= \beta^L (\mathcal{E}_{\rho, A}[f]) \sigma \quad \text{[(3.8)]} \\
= \beta^L (\mathcal{E}_{\rho_1 A_1}[f]) \sigma_1 \quad \text{[(3.2),(3.4),IH]} \\
= \mathcal{E}_{\rho_1 A_1}[(\text{elim } v(A s | tf) \mid f)] \sigma_1 \quad \text{[def.}\mathcal{E}\text{]}
\]

4.1. Let \(e \equiv (A s \mid f) t f \land s \neq t\)

4.2. Let \(A_2 \in \text{Assumptions} \; ; \; \rho_2 \in \text{Tenv} \; \text{and} \; \sigma_2 \in \text{ST}_{\rho_2, A_2}\) be such that
\(A_2 = A s \; \rho_2 = \rho[\mathcal{I}(tf) \rho / s] \; \sigma_2 = \sigma \uparrow \text{WTV}(A_2)\)

4.3. Let \(tf \in \text{Texp}\) be such that
\[\text{[(*)}, \text{ER8}]\]
a) \(\vdash A_2 : f : tf\)
b) \(te = \alpha^L tf^S_t f_1\)

4.4. \(\text{FTV}(e : te)\)
\[= \text{FTV}(e) \cup \text{FTV}(tf^S_{tf1})\]
\[= (\text{FTV}(f) \setminus \{s\}) \cup \text{FTV}(tf1) \cup (\text{FTV}(tf) \setminus \{s\})\]
\[= (\text{FTV}(f:tf) \setminus \{s\}) \cup \text{FTV}(tf1)\]

4.5. \(t \in \text{FTV}(f:tf)\) \[\text{[(**),(4.1),(4.4)]}\]

4.6. \(\rho_2[D/t] = \rho_1[\mathcal{I}(tf) \rho / s]\) \[\text{[s\neq t, def.}\rho_1, \rho_2]\]

4.7. \(\vdash A_2 : t : f : tf\) \[\text{[(4.3a),(4.5),ER11.1]}\]

4.8. \(\vdash A_1 : s : f : tf\) \[\text{[(4.7),ER12.1]}\]

4.9. Since \(\text{WTV}(A_2|t) = \text{WTV}(A_1|s)\) it follows that
\[(\sigma_2 \uparrow \text{WTV}(A_2|t)) \uparrow \text{WTV}(A_1|s) = (\sigma_1 \uparrow \text{WTV}(A_1|s)) \uparrow \text{WTV}(A_2|t)\]
4.10. \[ \varepsilon_{p,A}[([\Lambda s] f)\text{tf1}]\sigma \]
\[ = \varepsilon_{p_2,A_2}[f]\sigma_2 \]
\[ = \varepsilon_{p_2[D/t],A_2;\text{t}}[f]\sigma_2 \downarrow \text{WTV(A}_2;\text{t}) \]
\[ = \varepsilon_{p_1[\text{tf1}]\rho / s],A_2;\text{t}}[f]\sigma_2 \downarrow \text{WTV(A}_2;\text{t}) \]
\[ = \varepsilon_{p_1[\text{tf1}]\rho / s],A_1;\text{t}}[f]\sigma_1 \downarrow \text{WTV(A}_1;\text{t}) \]
\[ = \varepsilon_{p_1,A_1}[([\Lambda s] f)\text{tf1}]\sigma_1 \]

5.1. Let \( e = ([\Lambda t] f)\text{tf1} \)

5.2. Let \( tf \in \text{Texp} \) be such that
\[ ([*],\text{ER8.1}) \]
\[ \text{a)} \vdash A_1 \triangleright f : \text{tf} \]
\[ \text{b)} te \equiv \alpha^t_{\text{tf1}} \]

5.3. \( \vdash A_1;\text{t} \triangleright f : \text{tf} \)

5.4. Since \( \text{WTV(A}_1) = \text{WTV(A}_1;\text{t}) \) it follows that
\[ \vdash \sigma_1 \downarrow \text{WTV(A}_1;\text{t}) = (\sigma_1 \downarrow \text{WTV(A}_1;\text{t})) \downarrow \text{WTV(A}_1) \]

5.5. \( t \in \text{Ftv(tf1)} \)

5.6. \( \rho[\text{tf1}]\rho / t \)
\[ = \rho[D/t][\text{tf1}]\rho / t \]
\[ = \rho_1[\text{tf1}]\rho / t \]
\[ = \rho_1[\text{tf1}]\rho[D/t] / t \]
\[ = \rho_1[\text{tf1}]\rho_1 / t \]

5.7. \[ \varepsilon_{p,A}[([\Lambda t] f)\text{tf1}]\sigma \]
\[ = \varepsilon_{p[\text{tf1}]\rho / t],A_1\sigma_1 \]
\[ = \varepsilon_{p_1[\text{tf1}]\rho / t],A_1\sigma_1 \]
\[ = \varepsilon_{p_1[\text{tf1}]\rho / t],A_1;\text{t}}[f]\sigma_1 \downarrow \text{WTV(A}_1;\text{t}) \]
\[ = \varepsilon_{p_1,A_1}[([\Lambda t] f)\text{tf1}]\sigma_1 \]
CHAPTER 5

Theorem 5.3.2. [Substitution of type expressions for type variables in expressions]

For all $\rho \in \text{Tenv} ; A_1, A_2 \in \text{Assumptions} ; t \in \text{Tvar} ; te, te_1 \in \text{Texp} ; e \in \text{Exp}$ and $\sigma \in \text{ST}^t_{\rho, A_1 ; A_2}$:

If 
\[ \vdash A_1 \rightarrow te_1 \] 
\[ \vdash A_1 ; t ; A_2 \rightarrow e : te \] 
\[ \text{FTV}(A_1 ; t) \cap \text{FTV}(A_2) = \emptyset \]

Then 
\[ \mathcal{E}_{\rho, A_1 ; A_2 ; te_1}^t [e_{te_1}] = \mathcal{E}_{\rho_1, A_1 ; t ; A_2} [e] \sigma_1 \]

where $\rho_1 = \rho[\mathcal{F}[te_1] / t]$ and $\sigma_1 = \sigma[\text{WTV}(A_1 ; t ; A_2)]$

Proof. By induction on the structure of expression $e$. We prove only a limited number of difficult cases. Assume (*) ,(**) and (***) .

1.1. Let $e \equiv x$

1.2. $x \in \text{WTV}(A_1 ; t ; A_2)$

1.3. \[ \mathcal{E}_{\rho, A_1 ; A_2 ; te_1}^t [x_{te_1}] = \mathcal{E}_{\rho_1, A_1 ; t ; A_2} [x] \sigma \] \[\text{[subst]}\] 
\[\text{[def.$\mathcal{E}$]}\] 
\[\text{[(1.2)]}\] 
\[\mathcal{E}_{\rho_1, A_1 ; t ; A_2} [x] \sigma_1 \] \[\text{[def.$\mathcal{E}$]}\]

2.1. Let $e \equiv (\lambda y : ty \mid f)$

2.2. Let $tf \in \text{Texp}$ be such that

a) $\vdash A_1 ; t ; A_2 \rightarrow ty ; tf$

b) $\vdash A_1 ; t ; A_2 ; y : ty \rightarrow f : tf$

2.3. $t \in \text{FTV}(A_2 ; y : ty) = \text{FTV}(A_2) \] \[\text{[(***)]}\]

2.4. \[ \mathcal{F}[ty_{te_1}] \rho = \mathcal{F}[ty] \rho_1 \] \[\text{[(4.4.3)]}\]

2.5. Let $d \in \mathcal{F}[ty] \rho_1$. Moreover let $A_3 \in \text{Assumptions}$ and $\sigma_3 \in \text{ST}^t_{\rho_1, A_1 ; t ; A_3}$ be such that $A_3 = A_2 ; y : ty$

\[ \sigma_3 = \sigma_1[d/y] \]
2.6. \( \sigma[d/y] \vdash \text{WTV}(A_1;t;A_3) \)
\[= (\sigma \uparrow \text{WTV}(A_1;t;A_2))[d/y] \]
\[= \sigma_1[d/y] \] 
\[= \sigma_3 \] 

2.7. \( \varepsilon_{\rho,A_1;A_3}^t \left[ \Gamma_{\text{te}_1}^{t} \right] \sigma[d/y] \)
\[= \varepsilon_{\rho_1,A_1;A_3}^t [f](\sigma[d/y] \vdash \text{WTV}(A_1;t;A_3)) \)  
\[= \varepsilon_{\rho_1,A_1;A_3}^t [f]\sigma_3 \] 
\[= \varepsilon_{\rho,A_1;A_2}^t \left[ (\lambda y : ty \mid \Gamma_{\text{te}_1}^{t} ) \right] \sigma \] 
\[= \varepsilon_{\rho,A_1;A_2}^t \left[ (\lambda y : ty \mid \Gamma_{\text{te}_1}^{t} ) \right] \sigma \] 
\[= (\lambda d \in \mathcal{T}[\Gamma_{\text{te}_1}^{t}] \rho \mid \varepsilon_{\rho,A_1;A_3}^t \left[ \Gamma_{\text{te}_1}^{t} \right] \sigma ) \] 
\[= (\lambda d \in \mathcal{T}[\Gamma_{\text{te}_1}^{t}] \rho_1 \mid \varepsilon_{\rho_1,A_1;A_3}^t [f] \sigma_3 ) \] 
\[= \varepsilon_{\rho_1,A_1;A_2}^t \left[ (\lambda y : ty \mid f) \right] \sigma_1 \] 

3.1. Let \( e = (\text{elim } \nu(A s \mid t f) \mid f) \) 

3.2. \( \vdash A_1;t;A_2 \triangleright f : f s^t_{\nu(A s \mid t f)} \)  

3.3. Let \( r \) be the first type variable such that 
\[ r \neq t \land r \in \text{FTV}(t f) \land r \in \text{FTV}(te_1) \] 

3.4. Let \( (A,(\alpha^L,\alpha^R)) \) be the unique initial fixed point resulting from the inverse limit construction with functor \( \text{abstr}_r \mathcal{T}[(ts^1_{\text{te}_1})] \rho \) 

3.5. Let \( (B,(\beta^L,\beta^R)) \) be the unique initial fixed point resulting from the inverse limit construction with functor \( \text{abstr}_s \mathcal{T}[t f] \rho_1 \) 

3.6. \( (\text{abstr}_r \mathcal{T}[(ts^1_{\text{te}_1})] \rho = (\text{abstr}_s \mathcal{T}[t f]) \rho_1 \)  

3.7. \( \alpha^L = \beta^L \)  

3.8. \( \varepsilon_{\rho,A_1;A_2}^t \left[ (\text{elim } \nu(A s \mid t f) \mid f)_{\text{te}_1}^{t} \right] \sigma \)
\[= \varepsilon_{\rho,A_1;A_2}^t \left[ (\text{elim } \nu(A r \mid (ts^1_{\text{te}_1}) f_{\text{te}_1}^{t}) \right] \sigma \] 
\[= \varepsilon_{\rho,A_1;A_2}^t \left[ (\text{elim } \nu(A r \mid (ts^1_{\text{te}_1})) f_{\text{te}_1}^{t}) \right] \sigma \]
CHAPTER 5

4.1. Let \( e \equiv (A \, s \mid f)\text{tf}1 \)

4.2. Let \( tf \in \text{Texp} \) be such that

a) \( \vdash A_1; t; A_2; s \triangleright f : \text{tf} \)

b) \( te \equiv \alpha \, \text{tf}^s \)

4.3. Let \( r \) be the first type variable such that

\( r \not\equiv t \land r \not\in \text{FTV}(te_1) \land r \not\in \text{FTV}(ft; tf) \)

4.4. Let \( \rho_n \in \text{Tenv} \) and \( A_n \in \text{Assumptions} \) and \( \sigma_n \in \text{ST} \rho_n, A_n \), \( 2 \leq n \leq 5 \), be such that

\[
\begin{align*}
\rho_2 &= \rho_1[\mathcal{F}[\text{tf}1]\rho_1 / s] \quad \sigma_2 = \sigma_1 \uparrow \text{WTV}(A_1; t; A_2; s) \\
A_3 &= (A_2; r)^t_{te_1} \quad \rho_3 = \rho[\mathcal{F}[\text{tf}^t_{te_1}]\rho / r] \quad \sigma_3 = \sigma \uparrow \text{WTV}(A_1; A_3) \\
A_4 &= A_2; r \quad \rho_4 = \rho_3[\mathcal{F}[te_1]\rho_3 / t] \quad \sigma_4 = \sigma_3 \uparrow \text{WTV}(A_1; t; A_4) \\
A_5 &= A_4; s \quad \rho_5 = \rho_4[\mathcal{F}[r]\rho_4 / s] \quad \sigma_5 = \sigma_4 \uparrow \text{WTV}(A_1; t; A_5)
\end{align*}
\]

4.5. \( \vdash A_1; t; A_2; s; r \triangleright f : \text{tf} \) [(4.2a),(4.3),ER11.1]

4.6. \( \vdash A_1; t; A_4 \triangleright r \) [TR2]

4.7. \( \vdash A_1; t; A_5 \triangleright f : \text{tf} \) [(4.5),ER12.1]

4.8. \( \vdash A_1; t; A_4 \triangleright f^s_{r} : \text{tf}^s_{r} \) [(4.6),(4.7),ER14]

4.9. \( t \not\in \text{FTV}(A_2; r) \) [(***)/(4.3)]

4.10. \( \rho_4 \)

\[
\begin{align*}
\rho_4 &= \rho_5[\mathcal{F}[te_1]\rho_3 / t] \quad \text{[def.rho_4]} \\
\rho_4 &= \rho_5[\mathcal{F}[te_1]\rho_3[\mathcal{F}[\text{tf}^t_{te_1}]\rho / r] / t] \quad \text{[def.rho_5]} \\
\rho_4 &= \rho_5[\mathcal{F}[te_1]\rho / t] \quad \text{[(4.3),thm.4.4.1]} \\
\rho_4 &= \rho[\mathcal{F}[\text{tf}^t_{te_1}]\rho / r][\mathcal{F}[te_1]\rho / t] \quad \text{[def.rho_3]} \\
\rho_4 &= \rho[\mathcal{F}[\text{tf}1]\rho_1 / r][\mathcal{F}[te_1]\rho / t] \quad \text{[(4.4.3),def.rho_1]}
\end{align*}
\]

48
Theorem 5.3.3 [Renaming a bound type variable]

For all $\rho \in \text{Tenv}; A \in \text{Assumptions}; s, t \in \text{Tvar}; \text{te,te}_1 \in \text{Texp}; e \in \text{Exp} \text{ and } \sigma \in \text{ST} \rho, A$:

If

1. $\vdash A \triangleright \text{te}_1$  
   (*)&

2. $\vdash A; t \triangleright e; \text{te}$  
   (**) 

3. $s \in \text{FTV}(e; \text{te})$  
   (***)

\[\vdash \text{te}_1 \vdash (\rho, A)_{\text{te}_1} \triangleleft (\rho, A)_{\text{te}} \triangleleft (\rho, A)_{\text{te}}\]
CHAPTER 5

Then \[ \varepsilon_{\rho, A}[((\Lambda \tau | e) t) e \tau 1] \sigma = \varepsilon_{\rho, A}[((\Lambda s | e) s \tau 1) e \tau 1] \sigma \]

Proof. Assume (\ast),(\ast\ast) and (\ast\ast\ast).

1. Let \( \rho_1, \rho_2 \in \text{Tenv} ; A_1, A_2 \in \text{Assumptions} \) and \( \sigma_1 \in \text{ST} \rho_1, A_1, \sigma_2 \in \text{ST} \rho_2, A_2 \) be such that
   \[ A_1 = A; t \quad \rho_1 = \rho[\mathcal{F}[\tau 1] \rho / t] \quad \sigma_1 = \sigma \uparrow \text{WTV}(A_1) \]
   \[ A_2 = A; s \quad \rho_2 = \rho[\mathcal{F}[\tau 1] \rho / s] \quad \sigma_2 = \sigma \uparrow \text{WTV}(A_2) \]

2. \( \vdash A_2 \triangleright s \) \quad \text{[(1),TR2]}

3. \( \vdash A_1; s \triangleright e : t e \) \quad \text{[(1),\ast\ast,\ast\ast\ast,ER11.1]}

4. Since \( \text{FTV}(A) \subseteq \text{FTV}(A_2) \) it follows from (\ast) and prop.2.4.1 that
   \( \vdash A_2 \triangleright t e \)

5. \( \vdash A_2; t \triangleright e : t e \) \quad \text{[(3),ER12.1]}

6. \( \sigma_2 \uparrow \text{WTV}(A_2; t) \uparrow \text{WTV}(A_1; s) \)
   \[ = (\sigma \uparrow \text{WTV}(A_2; t)) \uparrow \text{WTV}(A_1; s) \]
   \[ = (\sigma \uparrow \text{WTV}(A_1; s)) \uparrow \text{WTV}(A_2; t) \]
   \[ = (\sigma_1 \uparrow \text{WTV}(A_1; s)) \uparrow \text{WTV}(A_2; t) \]

7. \[ \rho_2[\mathcal{F}[s] \rho_2 / t] \]
   \[ = \rho_2[\rho_2(s) / t] \quad \text{[def.\mathcal{F}]} \]
   \[ = \rho_2[\mathcal{F}[\tau 1] \rho / t] \quad \text{[(1)]} \]
   \[ = \rho[\mathcal{F}[\tau 1] \rho / s][\mathcal{F}[\tau 1] \rho / t] \quad \text{[(1)]} \]
   \[ = \rho[\mathcal{F}[\tau 1] \rho / t][\mathcal{F}[\tau 1] \rho / s] \quad \text{[(1)]} \]
   \[ = \rho_2[\mathcal{F}[\tau 1] \rho / s] \]

8. \[ \varepsilon_{\rho, A}[((\Lambda s | e) s \tau 1) e \tau 1] \sigma \]
   \[ = \varepsilon_{\rho_2, A_2}[e_s \tau 1] \sigma_2 \]
   \[ = \varepsilon_{\rho_2[\mathcal{F}[s] \rho_2 / t], A_2; t}[e] \sigma_2 \uparrow \text{WTV}(A_2; t) \]
   \[ = \varepsilon_{\rho_2[\mathcal{F}[s] \rho_2 / t], A_1; s}[e] \sigma_1 \uparrow \text{WTV}(A_1; s) \]
   \[ = \varepsilon_{\rho_1[\mathcal{F}[\tau 1] \rho / s], A_1; s}[e] \sigma_1 \uparrow \text{WTV}(A_1; s) \]
   \[ = \varepsilon_{\rho_1, A_1}[e] \sigma_1 \]

\[ \text{[(2),(5),thm5.3.2]} \]
\[ \text{[(3),(5),(6),prop.5.2.2]} \]
\[ \text{[(7)]} \]
\[ \text{[(\ast\ast),(\ast\ast\ast),thm5.3.1]} \]

50
Theorem 5.3.4. [State modification]

For all $A \in \text{Assumptions}; \rho \in \text{Tenv}; te, te1 \in \text{Texp}; x \in \text{Var}; e \in \text{Exp}; \sigma \in \text{ST}^\rho_A$ and $d_1 \in \mathcal{F}[te1]p$:

If $\vdash A \triangleright te1$  
$\vdash A \triangleright e : te$  
$x \notin \text{FV}(e)$

Then $\varepsilon^\rho_A[e] \sigma = \varepsilon^\rho_{A_1}[e] \sigma_1$

where $A_1 = A;x:te1$ and $\sigma_1 = \sigma[d_1/x]$

Proof. By induction to the structure of expression $e$. We prove only a limited number of difficult cases. Assume $(*)$ and $(**)$.

1.1. Let $e \equiv y$  

1.2. $y \neq x$  

1.3. $\varepsilon^\rho_A[y] \sigma = \sigma(y) = \sigma_1(y) = \varepsilon^\rho_{A_1}[y] \sigma_1$  

2.1. Let $e \equiv (\lambda y:te2 \mid f) \land x \neq y$  

2.2. Let $d_2 \in \mathcal{F}[te2]p$ and let $A_2 \in \text{Assumptions}$ and $\sigma_2 \in \text{ST}^\rho_{A_2}$ be such that $A_2 = A;y:te2$ and $\sigma_2 = \sigma[d_2/y]$  

2.3. Let $tf \in \text{Texp}$ be such that

a) $\vdash A \triangleright te2, tf$

b) $\vdash A_2 \triangleright f : tf$

c) $te \equiv \alpha te2 \rightarrow tf$

2.4. $\vdash A_2 \triangleright te1$  

2.5. $x \notin \text{FV}(f)$
CHAPTER 5

2.6. \[ \varepsilon_{\rho, A_2} [f] \sigma_2 = \varepsilon_{\rho, A_2; x: x_1} [f] \sigma_2 [d_f / x] \] [(2.4), (2.3), (2.5), IH]

2.7. \[ \vdash A_2; x: x_1 \triangleright f : t_f \] [(2.3), (2.4), ER11.2]

2.8. \[ \vdash A_1; y: x_2 \triangleright f : t_f \] [(2.5), ER12.2]

2.9. \[ \sigma_2 [d_1 / x] \triangleright \text{WTV} (A_1; y: x_2) = \sigma_1 [d_2 / y] \triangleright \text{WTV} (A_2; x: x_1) \] \[ x \neq y \]

2.10. \[ \varepsilon_{\rho, A} [\lambda y: x_2 \mid f] \sigma \]
\[ = (\lambda d_2 \in \mathcal{F}[x_2] \rho \mid \varepsilon_{\rho, A_2} [f] \sigma_2 ) \] [def. \( \varepsilon \)]
\[ = (\lambda d_2 \in \mathcal{F}[x_2] \rho \mid \varepsilon_{\rho, A_2; x: x_1} [f] \sigma_2 [d_f / x] ) \] [(2.6)]
\[ = (\lambda d_2 \in \mathcal{F}[x_2] \rho \mid \varepsilon_{\rho, A_1; y: x_2} [f] \sigma_1 [d_2 / y] ) \] [(2.7), (2.8), (2.9), prop. 5.2.2]
\[ = \varepsilon_{\rho, A_1} [\lambda y: x_2 \mid f] \sigma_1 \] [def. \( \varepsilon \)]

3.1. Let \( e \equiv (\lambda x: x_2 \mid f) \)

3.2. Let \( t_f \in \text{Texp} \) be such that \[ (**), \text{ER6.1} \]

a) \[ \vdash A \triangleright t_f , f \]

b) \[ \vdash A; x: x_2 \triangleright f : t_f \]

c) \[ t_e \equiv \alpha \ y \rightarrow t_f \] [(3.2b)]

3.3. \[ \vdash A_1; x: x_2 \triangleright f : t_f \] [(3.2b)]

3.4. \[ \sigma [d_f / x] \triangleright \text{WTV} (A_1; x: x_2) = \sigma_1 [d_f / x] \triangleright \text{WTV} (A_1; x: x_2) \]

3.5. \[ \varepsilon_{\rho, A} [\lambda y: y_2 \mid f] \sigma \]
\[ = (\lambda d \in \mathcal{F}[x_2] \rho \mid \varepsilon_{\rho, A_1; y: y_2} [f] \sigma [d_f / y] ) \] [def. \( \varepsilon \)]
\[ = (\lambda d \in \mathcal{F}[y_2] \rho \mid \varepsilon_{\rho, A_1; y: y_2} [f] \sigma_1 [d_f / y] ) \] [(3.3), (3.4), prop 5.2.2]
\[ = \varepsilon_{\rho, A_1} [\lambda y: y_2 \mid f] \sigma_1 \] [def. \( \varepsilon \)]

4.1. Let \( e \equiv (\Lambda s \mid f)t_f l \) [thm.5.3.3]

4.2. Assume without loss of generality that \( s \in \text{FTV} (t_f l) \) [(*)](4.2), ER11.1]

4.3. Let \( A_2 \in \text{Assumptions and } \sigma_2 \in \text{ST}_{\rho_2, A_2} \) be such that \( A_2 = A_1; s \quad \rho_2 = \rho [\mathcal{F}[t_f l] \rho / s] \quad \sigma_2 = \sigma \triangleright \text{WTV} (A_2) \)

4.4. \[ \vdash A_2 \triangleright t_e \] [(*)](4.2), ER11.1]
Let \( tf \in \text{Exp} \) be such that

\[ \vdash A_2 \rightarrow f : tf \]

\( x \notin \text{FV}(f) \)

\[ \vdash A_2 \vdash f : tf \]

\[ \vdash A_1 \vdash f : tf \]

\[ \sigma_2[d/x] \uparrow \text{WTV}(A_1;s) \]

\[ \quad = (\sigma \uparrow \text{WTV}(A_2))[d/x] \uparrow \text{WTV}(A_1;s) \]

\[ \quad = (\sigma[d/x] \uparrow \text{WTV}(A_2;x:te1)) \uparrow \text{WTV}(A_1;s) \]

\[ \quad = (\sigma_1 \uparrow \text{WTV}(A_2;x:te1)) \uparrow \text{WTV}(A_1;s) \]

\[ \quad = (\sigma_1 \uparrow \text{WTV}(A_1;s)) \uparrow \text{WTV}(A_2;x:te1) \]

\[ \epsilon_\rho \eta [(\Lambda s | f)tf1] \sigma \]

\[ = \epsilon_\rho_2 \eta_2 [f] \sigma_2 \]

\[ = \epsilon_\rho_2 \eta_2 \eta_1 [f] \sigma_2 [d/x] \]

\[ = \epsilon_\rho_2 \eta_2 \eta_1 [f] \sigma_1 \uparrow \text{WTV}(A_1;s) \]

\[ = \epsilon_\rho A_1 [(\Lambda s | f)tf1] \sigma_1 \] (def. \( \epsilon \))

\[ \Box \]

**Theorem 5.3.5.** (Substitution of expressions for variables in expressions)

For all \( \rho \in \text{Tenv} ; A \in \text{Assumptions} ; te1 \in \text{Exp} ; x \in \text{Var} ; e,e_1 \in \text{Exp} \) and \( \sigma \in \text{ST}_\rho A : \)

If

\[ \vdash A \triangleright e_1 : te1 \] (\( \ast \))

\[ \vdash A_1 \triangleright e : te \] (\( \ast \ast \))

Then

\[ \epsilon_\rho A_1 [e^x_{e_1}] \sigma = \epsilon_\rho A_1 [e] \sigma_1 \]

where \( A_1 = A; x:te1 \) and \( \sigma_1 = \sigma[\epsilon_\rho A_1 [e_1] \sigma / x] \)

**Proof.** By induction on the structure of expression \( e \). We prove only a few difficult cases.

Assume (\( \ast \)) and (\( \ast \ast \)).
1.1. Let \( e \equiv x \)

1.2. \[
\varepsilon_{\rho,A}[^{x}e_{1}]\sigma
= \varepsilon_{\rho,A}[e_{1}]\sigma
= \sigma[\varepsilon_{\rho,A}[e_{1}]\sigma / x] (x)
= \varepsilon_{\rho,A_{1}}[x]_{\sigma_{1}}
\]

2.1. Let \( e \equiv y \land y \neq x \)

2.2. \[
\varepsilon_{\rho,A}[y^{x}e_{1}]\sigma = \varepsilon_{\rho,A}[y]_{\sigma} = \sigma(y) = \sigma_{1}(y) = \varepsilon_{\rho,A_{1}}[y]_{\sigma_{1}}
\]

3.1. Let \( e \equiv (\lambda y:te2 | f) \)

3.2. Let \( z \) be the first variable such that \( z \neq x \land z \notin \text{FV}(f) \land z \notin \text{FV}(e_{1}) \)

3.3. Let \( D = \{te2\}_{\rho} \) and \( d \in D \)

3.4. Let \( A_{n} \in \text{Assumptions} \) and \( \sigma_{n} \in \text{ST}_{\rho,A_{n}} \), \( 2 \leq n \leq 5 \), be such that

\[
\begin{align*}
A_{2} &= A_{1};y:te2 & \sigma_{2} &= \sigma_{1}[d/y] \\
A_{3} &= A;z:te2 & \sigma_{3} &= \sigma[d/z] \\
A_{4} &= A_{3};x:tel & \sigma_{4} &= \sigma_{3}[\varepsilon_{\rho,A_{3}}[e_{1}]\sigma_{3} / x] \\
A_{5} &= A_{4};y:te2 & \sigma_{5} &= \sigma_{4}[\varepsilon_{\rho,A_{4}}[z]_{\sigma_{4}} / y]
\end{align*}
\]

3.5. \( \vdash A_{3} \rightarrow e_{1} : te1 \) \hspace{1cm} \((3.10) \text{ in thm.2.5.4}\)

3.6. \( \vdash A_{4} \rightarrow f^{y}_{z} : tf \) \hspace{1cm} \((3.8) \text{ in thm.2.5.4}\)

3.7. \( \vdash A_{4} \rightarrow z : te2 \) \hspace{1cm} \((3.7) \text{ in thm.2.5.4}\)

3.8. \( \vdash A_{5} \rightarrow f : tf \) \hspace{1cm} \((3.6) \text{ in thm.2.5.4}\)

3.9. \[
\varepsilon_{\rho,A_{3}}[e_{1}]\sigma_{3} = \varepsilon_{\rho,A_{1}}[e_{1}]\sigma
\]

3.10. \( \vdash A_{2};z:te2 \rightarrow f : tf \) \hspace{1cm} \((3.5) \text{ in thm.2.5.4}\)

3.11. \[
\begin{align*}
\sigma_{5}
&= \sigma_{4}[\varepsilon_{\rho,A_{4}}[z]_{\sigma_{4}} / y] \hspace{1cm} \text{[def.}\sigma_{5}] \\
&= \sigma_{4}[\sigma_{4}(z) / y] \hspace{1cm} \text{[def.}\varepsilon] \\
&= \sigma_{4}((\sigma[d/z][\varepsilon_{\rho,A_{3}}[e_{1}]\sigma_{3} / x])(z) / y) \hspace{1cm} \text{[(3.4)]}
\end{align*}
\]
Let \( e = (A; s) \mid f) t f l \)

Assume without loss of generality that

\[ s \in \text{FTV}(e1; te1) \]

Let \( A_2 \in \text{Assumptions} \rho_2 \in \text{Tenv} \) and \( \sigma_2 \in \text{ST}_{\rho, A_2} \) be such that

\[ A_2 = A; s \quad \rho_2 = \rho[\mathcal{T}[t f l] / s] \quad \sigma_2 = \sigma \uparrow \text{WTV}(A_2) \]

Let \( tf \in \text{Texp} \) be such that

\[ ([**], \text{ER8}) \]

\( \vdash A_1; s \triangleright f : tf \)

\( \vdash A_2 \triangleright e1 : te1 \)

\( \vdash A_2; x : te1 \triangleright f : tf \)

\( \sigma \uparrow \text{WTV}(A_2) \)

\( = (\sigma \uparrow \text{WTV}(A_2)) \uparrow \text{WTV}(A) \)

\( = \sigma_2 \uparrow \text{WTV}(A) \)
4.8. \( \text{WTV}(A_1;s) \subseteq \text{WTV}(A_2;x:te1) \)  
\[ \sigma_2[\varepsilon_{\rho,A_2}[e_1]_2 / x] \vdash \text{WTV}(A_1;s) \]  
\[ = \sigma_2[\varepsilon_{\rho,A}[e_1]_1 / x] \vdash \text{WTV}(A_1;s) \]  
\[ = ((\sigma \vdash \text{WTV}(A_2))[\varepsilon_{\rho,A}[e_1]_1 / x]) \vdash \text{WTV}(A_1;s) \]  
\[ = (\sigma[\varepsilon_{\rho,A}[e_1]_1 / x]) \vdash \text{WTV}(A_2;x:tx) \vdash \text{WTV}(A_1;s) \]  
\[ = (\sigma_1 \vdash \text{WTV}(A_2;x:tx)) \vdash \text{WTV}(A_1;s) \]  
\[ = \sigma_1 \vdash \text{WTV}(A_1;s) \]  
\[ (*)\)  
\[ (4.5),(4.7),\text{prop.5.2.2} \]  

4.9. \( \text{WTV}((\lambda x:te1) e) \)  
\[ = \varepsilon_{\rho,A}[(\lambda s f_1^x e_1)_1]_1 \]  
\[ = \varepsilon_{\rho,A}[(\lambda s f_1^x e_1)_1]_1 \]  
\[ = \varepsilon_{\rho_2,A_2}[f_1^x]_2 \]  
\[ = \varepsilon_{\rho_2,A_2;x:te1}[f_1]_2 \]  
\[ = \varepsilon_{\rho_2,A_1;x:te1}[(\lambda y:te1) e]_1 \]  
\[ = \text{WTV}(A_1;s) \]  
\[ = \text{WTV}(A_1;s) \]  
\[ (4.8) \]  

4.10. \( \varepsilon_{\rho,A}[(\lambda s f_1^x e_1)_1]_1 \)  
\[ = \varepsilon_{\rho,A}[(\lambda s f_1^x e_1)_1]_1 \]  
\[ = \varepsilon_{\rho_2,A_2}[f_1^x]_2 \]  
\[ = \varepsilon_{\rho_2,A_2;x:te1}[f_1]_2 \]  
\[ = \varepsilon_{\rho_2,A_1;x:te1}[(\lambda y:te1) e]_1 \]  
\[ = \text{WTV}(A_1;s) \]  
\[ (4.4),(4.6),(4.9),\text{prop5.2.2} \]  

\( \square \)

Renaming bound variables should and indeed does not alter the meaning of an expression.

**Theorem 5.3.6.**[Renaming a bound variable]  
For all \( \rho \in \text{Tenv} ; A \in \text{Assumptions} ; \sigma \in \text{ST}_{\rho,A} ; te,te1 \in \text{Texp} ; x,y \in \text{Var} \) and \( e \in \text{Exp} : \)

If  
\[ \vdash A \triangleright te1 \]  
\[ \vdash A;x:te1 \triangleright e \vdash te \]  
\[ y \notin \text{FV}(e) \]  

Then  
\[ \varepsilon_{\rho,A}[(\lambda x:te1) e]_1 \sigma = \varepsilon_{\rho,A}[(\lambda y:te1) e_1^x]_1 \sigma \]  
and  
\[ \varepsilon_{\rho,A}[(\lambda s x:te1) e]_1 \sigma = \varepsilon_{\rho,A}[(\lambda s y:te1) e_1^x]_1 \sigma \]  

**Proof.** Assume \((*)\),\((**)\) and \((***)\).
1. Let \( D = \mathcal{F}[te1] \rho \) and \( d \in D \)

2. Let \( A_n \in \text{Assumptions} \) and \( \sigma_n \in \text{ST}_{\rho, A_n}, \ 0 \leq n \leq 4 \), be such that

\[
\begin{align*}
A_1 &= A; x : te1 \\
\sigma_1 &= \sigma[d/x] \\
A_2 &= A; y : te1 \\
\sigma_2 &= \sigma[d/y] \\
A_3 &= A_2; x : te1 \\
\sigma_3 &= \sigma_2[\epsilon_{\rho, A_2}[y] \sigma_2 / x] \\
A_4 &= A_1; y : te1 \\
\sigma_4 &= \sigma_1[d/y]
\end{align*}
\]

3. \( \vdash A_1 \triangleright te1 \) \((\ast), \text{ER11.2}\)

4. \( \vdash A_2 \triangleright y : te1 \) \((\ast), \text{ER2}\)

5. \( \vdash A_4 \triangleright e : te \) \((\ast\ast), (\ast\ast\ast), (4), \text{ER11.2}\)

6. \( \vdash A_3 \triangleright e : te \) \((5), \text{ER12.2}\)

7. \[
\begin{align*}
\sigma_3 &= \sigma_2[\epsilon_{\rho, A_2}[y] \sigma_2 / x] \\
&= \sigma_2[\sigma_2(y) / x] \\
&= \sigma_2[d/x] \\
&= \sigma_1[d/y] \\
&= \sigma_4
\end{align*}
\]

8. \[
\begin{align*}
\epsilon_{\rho, A_2}[e^x_y] \sigma_2 \\
&= \epsilon_{\rho, A_3}[e] \sigma_3 \\
&= \epsilon_{\rho, A_4}[e] \sigma_4 \\
&= \epsilon_{\rho, A_1}[e] \sigma_1
\end{align*}
\]

9. \[
\begin{align*}
\epsilon_{\rho, A}[(\lambda y : te1 \ | \ e^x_y)] \sigma \\
&= (\lambda d \in D \ | \ \epsilon_{\rho, A_2}[e^x_y] \sigma_2) \\
&= (\lambda d \in D \ | \ \epsilon_{\rho, A_1}[e] \sigma_1) \\
&= \epsilon_{\rho, A}[(\lambda x : te1 \ | \ e)] \sigma
\end{align*}
\]

The case of strict \( \lambda \)-abstraction is proved similarly.

\( \square \)
6. SOUNDNESS OF REDUCTION

6.1. Introduction.

In chapter 3 we have introduced a set of reduction rules for expressions. Furthermore, we have shown that for expressions $e_1$ and $e_2$ such that $e_1 \rightarrow e_2$ their values $E_{\rho,A}[e_1] \sigma$ and $E_{\rho,A}[e_2] \sigma$ are members of the same domain $F[\tau_A(e_1)] \rho = F[\tau_A(e_2)] \rho$. In this chapter we prove that the reduction rules of chapter 3 are sound, i.e. reducing an expression yields an expression with the same value. In order to prove this result we need some elementary properties.

Property 6.1.1. [strictness]

For all $A \in$ Assumptions; $\rho \in$ Tenv; $\sigma \in$ ST$_{\rho,A}$; $t, t_1 \in$ Texp and $f \in$ Exp:

If $\vdash A \triangleright f : t \Theta t_1$

Then $(E_{\rho,A}[f] \sigma) (F[t] \rho) = F[t_1] \rho$.

\hfill □

Property 6.1.2. [normal form]

For all $A \in$ Assumptions; $\rho \in$ Tenv; $\sigma \in$ ST$_{\rho,A}$ and $e \in$ Exp:

If $e$ is in normal form and $(\forall x \in WTV(A) \mid \sigma(x) \neq F[\tau_A(x)] \rho)$

Then $E_{\rho,A}[e] \sigma \neq F[\tau_A(e)] \rho$.

\hfill □

6.2. Soundness.

Theorem 6.2.1. [soundness]

For all $A \in$ Assumptions; $\rho \in$ Tenv; $\sigma \in$ ST$_{\rho,A}$ and $e_1, e_2 \in$ WTE($A$):
If \( e_1 \rightarrow e_2 \) and 
\[(\forall x \in \text{WT}(A) \mid \sigma(x) \neq 1 \tau_A(x))\rho \]

Then \( E_{\rho,A}[e_1] \sigma = E_{\rho,A}[e_2] \sigma \)

Proof. It is sufficient to prove the soundness of each of the rules \( \nu \) thru \( \eta \). Apart from rules \( \sigma_5, \pi_4, \pi_5, \beta_1, \beta_2 \) and \( \beta_3 \) this is a trivial exercise using definition 5.2.1.

Rule \( \sigma_5 \): (sums \( f_1 f_2 ) \rightarrow f \), provided \( x \notin \text{FV}(f) \)

where 
\( f_1 = (\lambda x : t e_1 \mid (\text{appls } f (\text{inls } x \mid t e_2)) \)

and 
\( f_2 = (\lambda x : t e_2 \mid (\text{appls } f (\text{inrs } t e_1 \mid x)) \)

Assume \( x \notin \text{FV}(f) \)

1. The lefthandside of \( \sigma_5 \) is an element of \( \text{WTE}(A) \) iff
   a) \( \vdash A \rightarrow t e_1, t e_2 \)
   b) there exists a type expression \( t e \in \text{Texp} \) such that \( \vdash A \rightarrow f : t e_1 \otimes t e_2 \Theta t e \)

2. Let \( D_1 = \tau[te_1] \rho, D_2 = \tau[te_2] \rho \) and \( D = \tau[te] \rho \). Moreover, let 
   \( A_1 = A; x : t e_1 \) and \( A_2 = A; x : t e_2 \).

3. Since \( x \notin \text{FV}(f) \) it follows by (1), (***) and rule ER11.2 that \( \vdash A_1 \rightarrow f : (t e_1 \otimes t e_2) \Theta t e \) and \( \vdash A_2 \rightarrow f : (t e_1 \otimes t e_2) \Theta t e \).

Hence \( \tau[A_1(f)] \rho = \tau[A_2(f)] \rho = \tau[A(f)] \rho \).

4. For \( d \in D_1 : \)

   \[ E_{\rho,A_1}[(\text{appls } f (\text{inls } x \mid t e_2))] \sigma[d/x] \]
   \[ = E_{\rho,A_1}[f] \sigma[d/x] (E_{\rho,A_1}[(\text{inls } x \mid t e_2)] \sigma[d/x]) \]
   \[ = E_{\rho,A_1}[f] \sigma[d/x] <1, E_{\rho,A_1}[x] \sigma[d/x] >_{D_1 \oplus D_2} \]
   \[ = E_{\rho,A_1}[f] \sigma <1,d>_{D_1 \oplus D_2} \]
   \[ = E_{\rho,A_1}[f] \sigma <1, d >_{D_1 \oplus D_2} \]

\[(1a),(1b),(***),\text{thm5.3.4}] \]

59
Similarly one proves for \( d \in D_2 \):

\[
\mathcal{E}_{\rho,A_2}[((\text{apps } f (\text{inrs } te1 \mid x)))]\sigma[d/x]
\]

\[=
\mathcal{E}_{\rho,A}[f]\sigma(<2,d>D_1 \oplus D_2)
\]

5. \( \mathcal{E}_{\rho,A}[f_1]\sigma \)

\[= (\lambda d \in D_1
\]

\[| \text{if } d = \downarrow D_1 \rightarrow \downarrow D \]

\[| \text{if } d \neq \downarrow D_1 \rightarrow \mathcal{E}_{\rho,A_1}[((\text{apps } f (\text{inls } x \mid te2)))]\sigma[d/x]
\]

\[\text{fi}
\]

\[= (\lambda d \in D_1
\]

\[| \text{if } d = \downarrow D_1 \rightarrow \mathcal{E}_{\rho,A}[f]\sigma(<1,d>D_1 \oplus D_2)
\]

\[| \text{if } d \neq \downarrow D_1 \rightarrow \mathcal{E}_{\rho,A}[f]\sigma(<1,d>D_1 \oplus D_2)
\]

\[\text{fi}
\]

\[= (\lambda d \in D_1 \mid \mathcal{E}_{\rho,A}[f]\sigma(<1,d>D_1 \oplus D_2))
\]

Similarly one proves that

\( \mathcal{E}_{\rho,A}[f_2]\sigma = (\lambda d \in D_2 \mid \mathcal{E}_{\rho,A}[f]\sigma(<2,d>D_1 \oplus D_2)) \)

6. \( \mathcal{E}_{\rho,A}[(\text{sums } f_1 f_2)]\sigma \)

\[= (\lambda d \in D_1 \oplus D_2
\]

\[| \text{if } d = <1,d_1>D_1 \oplus D_2 \rightarrow (\mathcal{E}_{\rho,A}[f_1]\sigma)(d_1)
\]

\[| \text{if } d = <2,d_2>D_1 \oplus D_2 \rightarrow (\mathcal{E}_{\rho,A}[f_2]\sigma)(d_2)
\]

\[\text{fi}
\]
\[ (\lambda d \in D_1 \otimes D_2 \rightarrow E_{\rho,A}[f]_{\sigma}(d)) \]

Rule \( \pi_4 : \text{(prods (prods e1 e2))} \rightarrow e1 \), provided \( e2 \) in normal form

Assume \( e2 \) is in normal form.

\[(\forall x \in WTV(A) \mid \sigma(x) \neq \mathcal{T}[\tau_A(e)_{\rho})\]

1. Let \( D_1 = \mathcal{T}[\tau_A(e1)_{\rho} \) and \( D_2 = \mathcal{T}[\tau_A(e2)_{\rho}\)

2. \( E_{\rho,A}[(\text{prods (prods e1 e2))}]_{\sigma} = \psi(\mathcal{T}[e1]_{\sigma}, E_{\rho,A}[e2]_{\sigma} > D_1 \otimes D_2) \)

\[ = \text{if } E_{\rho,A}[e2]_{\sigma} = \mathcal{T}[e2]_{\sigma} \rightarrow \mathcal{T}[e1]_{\sigma}
\]

Similarly one proves the soundness of rule \( \pi_5 \).

Rules \( \beta_1, \beta_2, \beta_3 : \)

The soundness of rules \( \beta_1 \) and \( \beta_2 \) follows from a simple computation using theorem 5.3.2.

The soundness of rule \( \beta_3 \) follows from theorem 5.3.1.
CHAPTER 7

7.A TYPED FIXED POINT COMBINATOR

7.1 Syntax.

In the type free lambda calculus every term can be considered as a function. Moreover, every term (function) has a fixed point which can be computed using a fixed point combinator. The most well known fixed point combinator in the type free lambda calculus is the Curry combinator

\[ \mu = (\lambda f \mid (\lambda x \mid f(xx)) (\lambda x \mid f(xx)) ) \]

A simple calculation shows that for every term \( g \) the terms \( \mu g \) and \( g(\mu g) \) are convertible; so \( \mu g \) can be considered as a fixed point of \( g \). It can be shown that in the \( D_\infty \) model of the type free lambda calculus \( \mu \) corresponds to the least fixed point operator, see for instance Wadsworth [Wa76].

In this chapter we show that similar results hold for the typed language described in this report. Let \( te \) be an arbitrary type expression. In this section we shall describe an expression \( (\mu \mid te) \) with type \( (te \rightarrow te) \rightarrow te \), which can be considered as a typed version of the Curry combinator. In \( (\mu \mid te) \) a recursively defined type will be used. Some properties associated with the corresponding domain (found by the inverse limit construction) are given in section 7.2. Finally in section 7.3 we show that in the appropriate domain \( (\mu \mid te) \) corresponds to the least fixed point operator.

In this chapter we use the following abbreviations

\[ w = \nu(\Lambda t \mid t \rightarrow te) \]

where \( t \) is the first type variable such that \( t \notin FTV(te) \) and

\[ g = (\lambda x : w \mid (\text{appl} f (\text{appl} (\text{intro} w \mid x) x)) ) . \]

A typed version of the Curry combinator is then given by

\[ (\mu \mid te) = ( f : te \rightarrow te \mid (\text{appl} g (\text{elim} w \mid g)) ) . \]

It is an elementary exercise to show that the following type inference rule holds
A \rightarrow te
A \rightarrow (\mu \mid te) : (te \rightarrow te) \rightarrow te

The following theorem states that $\mu \mid te$ is a syntactic fixed point combinator.

**Theorem 7.1.4.**

Let $f : te \rightarrow te$. Then $(\text{appl} \ (\mu \mid te) \ f)$ and $(\text{appl} \ f \ (\text{appl} \ (\mu \mid te) \ f))$ have a common reduct.

**Proof.** The theorem is easily proved by the following computations.

\[
(\text{appl} \ (\mu \mid te) \ f)
\]
\[
\quad \rightarrow (\text{appl} \ g \ (\text{elim} \ w \mid g)) \quad [\beta_1]
\]
\[
\quad \rightarrow (\text{appl} \ (\lambda \ x : w \mid (\text{appl} \ f \ (\text{appl} \ (\text{intro} \ w \mid x)) \ x)) \ (\text{elim} \ w \mid g)) \quad [(7.1.2)]
\]
\[
= (\text{appl} \ f \ (\text{appl} \ (\text{intro} \ w \mid (\text{elim} \ w \mid g)) \ (\text{elim} \ w \mid g))) \quad [\beta_1]
\]
\[
\quad \rightarrow (\text{appl} \ f \ (\text{appl} \ g \ (\text{elim} \ w \mid g))) \quad [\varepsilon_2]
\]

Also

\[
(\text{appl} \ f \ (\text{appl} \ (\mu \mid te) \ f))
\]
\[
\quad \rightarrow (\text{appl} \ f \ (\text{appl} \ g \ (\text{elim} \ w \mid g))) \quad [\beta_1]
\]

which proves the theorem.

\[\square\]

Note that, although $(\text{appl} \ (\mu \mid te) \ f)$ and $(\text{appl} \ f \ (\text{appl} \ (\mu \mid te) \ f))$ have a common reduct, it is not possible to reduce one of these terms to the other. The same property holds for the untyped Curry combinator. For the untyped lambda calculus there exists another fixed point combinator, the Turing combinator $\mu'$, such that $\mu' f$ reduces to $f(\mu' f)$. A typed version of the Turing combinator, with similar properties as described in this report, can also be given, see for instance Struik[St88].
7.2. Technical results.

In the construction of \((\mu | te)\) we used the recursive type \(w = v(\Lambda t | t \to te)\). In a type environment \(\rho\) the corresponding domain \(W\) is obtained in the following way. Let the functor \(F: C \to C\) be given by

\[
F = (\text{abstr}_t F[t \to te])(\rho)
\]

Then, following the semantics of type expressions as described in section 4, we get

\[
W = \mathcal{F}[v(\Lambda t | t \to te)]\rho = \text{IFP}(F).
\] (7.2.1)

A simple computation (using the definition of \(\text{abstr}_t\) given in section 4.1) yields that

\[
F = \text{FS} \circ <I, C_B> : C \to C
\]

where \(I, C_B : C \to C\) are respectively the identity functor and the constant functor corresponding to the domain \(B = \mathcal{F}[te]\rho\). Recall that in the category \(C = \text{CPO}_{\text{PR}}\) a morphism \(\alpha \in \text{Hom}(A_1, A_2)\) is a pair \(\alpha = (\alpha^L, \alpha^R)\), where \(\alpha^L : A_1 \to A_2\) is an embedding and \(\alpha^R : A_2 \to A_1\) is a projection, i.e.

\[
\alpha^L \circ \alpha^R \subseteq \text{id}_{A_2} \quad \text{and} \quad \alpha^R \circ \alpha^L = \text{id}_{A_1} \quad (7.2.2)
\]

If \(\alpha\) is an isomorphism, then in the first relation equality holds and

\[
(\alpha^{-1})^L = \alpha^R \quad \text{and} \quad (\alpha^{-1})^R = \alpha^L.
\]

The composition of the morphisms \(\alpha = (\alpha^L, \alpha^R) \in \text{Hom}(A_1, A_2)\) and \(\beta = (\beta^L, \beta^R) \in \text{Hom}(A_2, A_3)\) is given by

\[
\beta \circ \alpha = (\beta^L \circ \alpha^L, \alpha^R \circ \alpha^L).
\]

From the definition of the function space functor \(\text{FS}\) (see for instance [BH88], where this functor is called \(A\)) it follows that (recall \(B = \mathcal{F}[te]\rho\))

- if \(A \in \text{obj}(C)\), then \(F(A) = [A \to B]\),
- if \(\alpha \in \text{Hom}(A_1, A_2)\), then \(F(\alpha) \in \text{Hom}( [A_1 \to B], [A_2 \to B] )\) is defined by

\[
F(\alpha)^L(\xi) = \xi \circ \alpha^R \quad \text{for} \quad \xi \in [A_1 \to B],
\] (7.2.3)

64
The object $W$ is constructed in the following way. Let $D_0$ be the initial object in the category $C$, i.e. $D_0$ is the one point c.p.o. Let $D_k = F^k(D_0)$ for $k \geq 1$. Since $D_0$ is initial, there exists a unique morphism $\psi_0 \in \text{Hom}(D_0, D_1)$. Let $\psi_k = F^k(\psi_0)$ for $k \geq 1$. Then $\Delta = \langle (D_i, \psi_i) \rangle_{i=0}^{\infty}$ is an $\omega$-chain in $C$. Since $C$ is an $\omega$-category this $\omega$-chain has a colimit $(W, \alpha)$. This defines the (an) object $W$. To see that $W$ is a fixed point of the functor $F$, we consider the $\omega$-chain $\Delta' = \langle (F(D_i), F(\psi_i)) \rangle_{i=0}^{\infty} = \langle (D_{i+1}, \psi_{i+1}) \rangle_{i=0}^{\infty}$. Since $F$ is an $\omega$-continuous functor it preserves colimits. So $(F(W), F(\alpha))$ (where $F(\alpha)$ stands for $\langle F(\alpha_i) \rangle_{i=0}^{\infty}$) is a colimit of the $\omega$-chain $\Delta'$. Apart from the first element of $\Delta$, the chains $\Delta$ and $\Delta'$ are identical. Thus $(W, \alpha)$ and $(F(W), F(\alpha))$ are both colimits of the same $\omega$-chain, which implies that there exists an isomorphism $\beta \in \text{Hom}(F(W), W)$. The situation may be elucidated by the following figure.

Now the following properties hold (see for instance Smyth and Plotkin[SP82] or Bos and Hemerik [BH88]).

\begin{align*}
\alpha_k^L \circ \alpha_k^R & \subseteq \text{id}_W \\
\alpha_k^L \circ \alpha_k^R & \subseteq \text{id}_{D_k} \\
\beta^{-1} \circ \alpha_{k+1} & = F(\alpha_k) \\
\beta \circ F(\alpha_k) & = \alpha_{k+1}
\end{align*}
CHAPTER 7

\[ \bigcup_{k=0}^{\infty} \alpha_k^L \circ \alpha_k^R = \text{id}_W \]  
(7.2.8)

Define the mapping \( P_k : W \to W \) by \( P_k = \alpha_k^L \circ \alpha_k^R \). Then (7.2.4) and (7.2.8) can be written as

\[ P_k \subseteq \text{id}_W \]  
(7.2.9)

and

\[ \bigcup_{k=0}^{\infty} P_k = \text{id}_W \]  
(7.2.10)

Since \( D_0 \) is the one–point c.p.o. and \( \alpha_0^L \) is strict, we have

\[ P_0(x) = \bot_W \]  
(7.2.11)

for all \( x \in W \).

In the remainder of this section we give some technical lemmas, which will be used in section 7.3.

Lemma 7.2.12.

If \( \ell \geq k \geq 0 \) then \( \alpha_k^R \circ P_\ell = \alpha_k^R \) and \( P_\ell \circ \alpha_k^L = \alpha_k^L \).

Proof. We prove the first relation for fixed \( \ell \) by induction with respect to \( k \). If \( k = \ell \) the result follows from (7.2.5). Next suppose \( \alpha_k^R \circ P_\ell = \alpha_k^R \) and \( \ell \geq k \geq 1 \). Since \((W, \alpha)\) is a cocone for \( \Delta \), we have \( \alpha_{k-1} = \alpha_k \circ \psi_{k-1} \), so \( \alpha_{k-1}^R = \psi_{k-1}^R \circ \alpha_k^R \). Then using the induction hypothesis, we get

\[ \alpha_{k-1}^R \circ P_\ell = \psi_{k-1}^R \circ \alpha_k^R \circ P_\ell = \psi_{k-1}^R \circ \alpha_k^R = \alpha_{k-1}^R \] .

The second part of the lemma can be proved in a similar way.

Lemma 7.2.13.

If \( k \geq 0 \) and \( \ell \geq 0 \) then \( P_k \circ P_\ell = P_{\min(k, \ell)} \).

Proof. The lemma follows immediately from the definition of \( P_k \) and lemma 7.2.12.

\( \square \)
Note that if \( y \in W \), then \( \beta^R(y) \in F(W) = [W \rightarrow B] \). In the case that \( y \in P_{k+1}(W) \) the mapping \( \beta^R(y) : [W \rightarrow B] \) has a special property.

**Lemma 7.2.14.**

Let \( x \in W \) and \( \ell \geq k \geq 0 \) . Then

\[
\beta^R(P_{k+1}(x)) = \beta^R(P_{k+1}(x)) \circ P_\ell.
\]

**Proof.** The lemma follows from the following computation.

\[
\begin{align*}
\beta^R(P_{k+1}(x)) &= F(\alpha^R_k) \circ (\alpha^R_{k+1}(x)) \quad \text{[L component of (7.2.6), } (\beta^{-1})^L = \beta^R] \\
&= (\alpha^R_{k+1}(x)) \circ \alpha^R_k \quad \text{[(7.2.3)]} \\
&= (\alpha^R_{k+1}(x)) \circ \alpha^R_k \circ P_\ell \quad \text{[lemma.7.2.12]}
\end{align*}
\]

\( \square \)

**7.3. Semantics.**

We now show that the semantics of \((\mu | te)\) is the least fixed point operator in the appropriate c.p.o. The computation given here, is similar to the computation of the semantics of the untyped Curry fixed point combinator as given in Wadsworth [Wa76].

Suppose that \( \vdash A \triangleright te \) and let \( \rho \in \text{Tenv} \). We introduce the following abbreviations:

\[
\begin{align*}
w &= \nu(\Lambda t | t \rightarrow te), \\
g &= (\lambda x:w | (\text{appl f (appl (intro w | x)) x})) , \\
W &= \mathcal{F}[\nu(\Lambda t | t \rightarrow te)] \rho , \\
B &= \mathcal{F}[te] \rho , \\
\chi &= A_1[g] \sigma[\phi/f] \quad \text{where } A_1 = A;f:te \rightarrow te
\end{align*}
\]
From the definition of the semantics of expressions, see [def. 5.2.1 case 7], it follows that

\[ \chi = E_{p,A}[\mu][f] = (\lambda d \in W \mid \phi((\beta^R(d)) d)) \]  \hspace{2cm} (7.3.1)

where \( \beta \) is the isomorphism between \( F(W) = \{W \rightarrow B\} \) and \( W \). From (7.1.3) we get

\[ E_{p,A}[\mu][t] = (\lambda \phi \in \{B \rightarrow B\} \mid \chi((\beta^L(\chi))) ) \]  \hspace{2cm} (7.3.2)

The following theorem shows that \( E_{p,A}[\mu][t] \) is a fixed point operator for the domain \( B \).

Theorem 7.3.3.

Let \( \phi \in \{B \rightarrow B\} \). Then for all states \( \sigma \in ST_{p,A} \)

\[ (E_{p,A}[\mu][t] \sigma) \phi = \phi ((E_{p,A}[\mu][t] \sigma) \phi) \]

Proof. The theorem follows from the following computation.

\[
\begin{align*}
( E_{p,A}[\mu][t] \sigma ) \phi \\
= \chi((\beta^L(\chi))) \\
= \phi((\beta^R((\beta^L(\chi)))) ) \\
= \phi(\chi((\beta^L(\chi))) ) \\
= \phi((E_{p,A}[\mu][t] \sigma) \phi) \\
\end{align*}
\]

\( \square \)

Since

\[ \mu_B = (\lambda \phi \in \{B \rightarrow B\} \mid \bigcup_{k=0}^{\infty} \phi^k(\mu_B) ) \]  \hspace{2cm} (7.3.4)

is the least fixed point operator in the c.p.o. \( B \), we now have

\[ \mu_B \subseteq E_{p,A}[\mu][t] \sigma . \]  \hspace{2cm} (7.3.5)

The following theorem shows that in (7.3.5) equality holds.
Theorem 7.3.6.

For all states $\sigma \in ST_{\rho,A}$,

$$\mathcal{E}_{\rho,A}[\mu|te]\sigma = \mu_B.$$ 

Proof. We first show by induction with respect to $k$ that for $k \geq 0$

$$\chi(P_k(\beta^L(\chi))) \subseteq \phi^{k+1}(\mu_B) \quad (7.3.7)$$

- Induction basis, $k = 0$. Then

$$\chi(P_0(\beta^L(\chi)))$$

$$= \chi(\mu_W) \quad [(7.2.11)]$$

$$= \phi(\beta^R(\mu_W)) \quad [(7.3.1)]$$

$$= \phi(\mu_B). \quad [\beta^R \text{ is strict}]$$

- Induction step. Suppose (7.3.7) holds. Then

$$\chi(P_{k+1}(\beta^L(\chi)))$$

$$= \phi(\beta^R(P_{k+1}(\beta^L(\chi))) \circ P_k(\beta^L(\chi))) \quad [(7.3.1)]$$

$$= \phi(\beta^R(P_{k+1}(\beta^L(\chi))) \circ P_k(\beta^L(\chi))) \quad [\text{lemma 7.2.14 with } \ell = k \text{ and } x = \beta^L(\chi)]$$

$$\subseteq \phi(\chi(P_k(\beta^L(\chi)))) \quad [(7.2.9), \beta^R \text{ and } \phi \text{ are monotonic}, (7.2.2)]$$

$$\subseteq \phi(\phi^{k+1}(\mu_B)) \quad [\text{induction hypothesis (7.3.7), } \phi \text{ is monotonic}]$$

This proves (7.3.7) for all $k \geq 0$. The theorem now follows from (7.3.5) and the following computation.

$$\mathcal{E}_{\rho,A}[\mu|te]\sigma$$

$$= (\lambda \phi \in [B \rightarrow B] | \chi(\bigcup_{k=0}^{\infty} P_k(\beta^L(\chi))) \quad [(7.3.2),(7.2.10)]$$
Thus we have shown that the semantics of \((\mu|\text{te})\) is the least fixed point operator in the domain \(B\) corresponding to the type expression \(\text{te}\). This result means that it is not necessary to add recursion explicitly to the language given in chapters 1 and 2. Recursion can be performed using the typed fixed point combinator \((\mu|\text{te})\), which can be written in terms of the already defined language. Note that the presence of recursively defined types is essential for the construction of \((\mu|\text{te})\).
REFERENCES


REFERENCES


In this series appeared:

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>85/01</td>
<td>R.H. Mak</td>
<td>The formal specification and derivation of CMOS-circuits</td>
</tr>
<tr>
<td>85/02</td>
<td>W.M.C.J. van Overveld</td>
<td>On arithmetic operations with M-out-of-N-codes</td>
</tr>
<tr>
<td>85/03</td>
<td>W.J.M. Lemmens</td>
<td>Use of a computer for evaluation of flow films</td>
</tr>
<tr>
<td>85/04</td>
<td>T. Verhoeff, H.M.J.L. Schols</td>
<td>Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate</td>
</tr>
<tr>
<td>86/01</td>
<td>R. Koymans</td>
<td>Specifying message passing and real-time systems</td>
</tr>
<tr>
<td>86/02</td>
<td>G.A. Bussing, K.M. van Hee, M. Voorhoeve</td>
<td>ELISA, A language for formal specifications of information systems</td>
</tr>
<tr>
<td>86/03</td>
<td>Rob Hoogerwoord</td>
<td>Some reflections on the implementation of trace structures</td>
</tr>
<tr>
<td>86/04</td>
<td>G.J. Houben, J. Paredaens, K.M. van Hee</td>
<td>The partition of an information system in several parallel systems</td>
</tr>
<tr>
<td>86/05</td>
<td>Jan L.G. Dietz, Kees M. van Hee</td>
<td>A framework for the conceptual modeling of discrete dynamic systems</td>
</tr>
<tr>
<td>86/06</td>
<td>Tom Verhoeff</td>
<td>Nondeterminism and divergence created by concealment in CSP</td>
</tr>
<tr>
<td>86/07</td>
<td>R. Gerth, L. Shira</td>
<td>On proving communication closedness of distributed layers</td>
</tr>
<tr>
<td>86/09</td>
<td>C. Huizing, R. Gerth, W.P. de Roever</td>
<td>Full abstraction of a real-time denotational semantics for an OCCAM-like language</td>
</tr>
<tr>
<td>86/10</td>
<td>J. Hooman</td>
<td>A compositional proof theory for real-time distributed message passing</td>
</tr>
<tr>
<td>86/11</td>
<td>W.P. de Roever</td>
<td>Questions to Robin Milner - A responder's commentary (IFIP86)</td>
</tr>
<tr>
<td>86/12</td>
<td>A. Boucher, R. Gerth</td>
<td>A timed failures model for extended communicating processes</td>
</tr>
</tbody>
</table>
86/13 R. Gerth  
W.P. de Roever  
Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)

86/14 R. Koymans  
Specifying passing systems requires extending temporal logic

87/01 R. Gerth  
On the existence of sound and complete axiomatizations of the monitor concept

87/02 Simon J. Klaver  
Chris F.M. Verberne  
Federatieve Databases

87/03 G.J. Houben  
J.Paredaens  
A formal approach to distributed information systems

87/04 T. Verhoeff  
Delay-insensitive codes - An overview

87/05 R. Kuiper  
Enforcing non-determinism via linear time temporal logic specification.

87/06 R. Koymans  
Temporele logica specificatie van message passing en real-time systemen (in Dutch).

87/07 R. Koymans  
Specifying message passing and real-time systems with real-time temporal logic.

87/08 H.M.J.L. Schols  
The maximum number of states after projection.

87/09 J. Kalisvaart  
L.R.A. Kessener  
W.J.M. Lemmens  
M.L.P. van Lierop  
F.J. Peters  
H.M.M. van de Wetering  
Language extensions to study structures for raster graphics.

87/10 T. Verhoeff  
Three families of maximally nondeterministic automata.

87/11 P. Lemmens  
Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.

87/12 K.M. van Hee and A. Lapinski  
OR and AI approaches to decision support systems.

87/13 J.C.S.P. van der Woude  
Playing with patterns, searching for strings.

87/14 J. Hooman  
A compositional proof system for an occam-like real-time language
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>87/15</td>
<td>C. Huizing, R. Gerth, W.P. de Roever</td>
<td>A compositional semantics for statecharts</td>
</tr>
<tr>
<td>87/16</td>
<td>H.M.M. ten Eikelder, J.C.F. Wilmont</td>
<td>Normal forms for a class of formulas</td>
</tr>
<tr>
<td>87/17</td>
<td>K.M. van Hee, G.-J. Houben, J.L.G. Dietz</td>
<td>Modelling of discrete dynamic systems framework and examples</td>
</tr>
<tr>
<td>87/18</td>
<td>C.W.A.M. van Overveld</td>
<td>An integer algorithm for rendering curved surfaces</td>
</tr>
<tr>
<td>87/19</td>
<td>A.J. Seebregts</td>
<td>Optimisering van file allocatie in gedistribueerde database systemen</td>
</tr>
<tr>
<td>87/20</td>
<td>G.J. Houben, J. Paredaens</td>
<td>The $R^2$-Algebra: An extension of an algebra for nested relations</td>
</tr>
<tr>
<td>87/21</td>
<td>R. Gerth, M. Codish, Y. Lichtenstein, E. Shapiro</td>
<td>Fully abstract denotational semantics for concurrent PROLOG</td>
</tr>
<tr>
<td>88/01</td>
<td>T. Verhoeff</td>
<td>A Parallel Program That Generates the Möbius Sequence</td>
</tr>
<tr>
<td>88/02</td>
<td>K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve</td>
<td>Executable Specification for Information Systems</td>
</tr>
<tr>
<td>88/03</td>
<td>T. Verhoeff</td>
<td>Settling a Question about Pythagorean Triples</td>
</tr>
<tr>
<td>88/04</td>
<td>G.J. Houben, J. Paredaens, D. Tahon</td>
<td>The Nested Relational Algebra: A Tool to handle Structured Information</td>
</tr>
<tr>
<td>88/05</td>
<td>K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve</td>
<td>Executable Specifications for Information Systems</td>
</tr>
<tr>
<td>88/06</td>
<td>H.M.J.L. Schols</td>
<td>Notes on Delay-Insensitive Communication</td>
</tr>
<tr>
<td>88/07</td>
<td>C. Huizing, R. Gerth, W.P. de Roever</td>
<td>Modelling Statecharts behaviour in a fully abstract way</td>
</tr>
<tr>
<td>88/08</td>
<td>K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve</td>
<td>A Formal model for System Specification</td>
</tr>
<tr>
<td>88/09</td>
<td>A.T.M. Aerts, K.M. van Hee</td>
<td>A Tutorial for Data Modelling</td>
</tr>
<tr>
<td>Issue</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>88/10</td>
<td>J.C. Ebergen</td>
<td>A Formal Approach to Designing Delay Insensitive Circuits</td>
</tr>
<tr>
<td>88/11</td>
<td>G.J. Houben, J.Paredaens</td>
<td>A graphical interface formalism: specifying nested relational databases</td>
</tr>
<tr>
<td>88/12</td>
<td>A.E. Eiben</td>
<td>Abstract theory of planning</td>
</tr>
<tr>
<td>88/13</td>
<td>A. Bijlsma</td>
<td>A unified approach to sequences, bags, and trees</td>
</tr>
<tr>
<td>88/14</td>
<td>H.M.M. ten Eikelder, R.H. Mak</td>
<td>Language theory of a lambda-calculus with recursive types</td>
</tr>
<tr>
<td>88/15</td>
<td>R. Bos, C. Hemerik</td>
<td>An introduction to the category theoretic solution of recursive domain equations</td>
</tr>
<tr>
<td>88/16</td>
<td>C. Hemerik, J.P. Katoen</td>
<td>Bottom-up tree acceptors</td>
</tr>
<tr>
<td>88/17</td>
<td>K.M. van Hee, G.J. Houben, L.J. Somers, M. Voorhoeve</td>
<td>Executable specifications for discrete event systems</td>
</tr>
</tbody>
</table>