A formal account of notational definition

Timothy G. Griffin*

88-901
March 1988

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported in part by NSF grant no. MCS-83-03327. An abbreviated version of this paper will appear in the *Proceedings of the Third Symposium on Logic in Computer Science,* IEEE, July 1988.

A formal account of notational definition

Timothy G. Griffin*
Cornell University

March 15, 1988

Abstract

In the course of developing a mathematical theory or proof it is a common practice to introduce new notation to represent notation that is previously understood. This paper presents a formal account that is intended to model the practice of introducing and using notational (abbreviative) definitions. The aim of this work is a pragmatic one: to provide a framework useful in the design and implementation of secure proof system interfaces which accommodate, as much as possible, conventional mathematical practice. A typed λ-calculus is used to represent expressions of a given object language. A new type of equation, called a Δ-equation, is introduced to model conventional definitional equations.

*This work was supported in part by NSF grant no. MCS-83-03327. An abbreviated version of this paper will appear in the *Proceedings of the Third Symposium on Logic in Computer Science*, IEEE, July 1988.

Contents

1 Introduction. **3**
- 1.1 Motivation. . . . 3
- 1.2 An overview of this report. . . . 6

2 A typed λ-calculus. **7**
- 2.1 Syntax. . . . 7
- 2.2 Typing expressions. . . . 9
- 2.3 Reduction and definitional equality. . . . 11

3 Representing languages à la Church. **12**

4 Δ-equations. **14**
- 4.1 Informal outline. . . . 14
- 4.2 Making it precise. . . . 17
- 4.3 Acceptable expansions. . . . 24

5 The expansion Exp^{Δ}. **24**
- 5.1 Generalized substitution. . . . 25
- 5.2 Simultaneous α-conversion. . . . 26
- 5.3 Exp^{Δ}. . . . 27

6 Implementation considerations. **29**
- 6.1 Implementing $\mathcal{L}$. . . . 29
- 6.2 Declaration of defined notation. . . . 30

7 Related work. **30**
- 7.1 Lisp macros. . . . 30
- 7.2 Nuprl. . . . 31
- 7.3 EFS. . . . 31

8 Acknowledgments. **32**

9 Appendix **32**

A Technical results. **32**

1 Introduction.

In the course of developing a mathematical theory or proof it is a common practice to introduce new notation to represent notation that is previously understood. Indeed, the definition of abbreviative notation is so much a part of mathematical practice that is difficult to imagine "doing mathematics" without it. A formal account is presented here that is intended to model this practice of introducing and using notational definitions. It is doubtful that any one formalism could capture all of the various contrivances employed by mathematicians, most often informally, when fashioning definitions. Still, we believe that the account presented here encompasses a large and useful class of abbreviative definitions.

The aim of this work is a pragmatic one: to provide a framework useful in the design and implementation of secure proof system interfaces which accommodate, as much as possible, conventional mathematical practice. In particular, we have in mind interactive systems that support formal reasoning, allowing users to state mathematical assertions, to construct formal proofs of these assertions in some logic, and to organize results. Examples of such systems include AUTOMATH [dB70,dB80], the LCF project [GMW79,Pau85], the Interactive Proof Editor (IPE) [Bur86,Jon87], the Göteborg group's implementation of type theory [Pet82,NPS86], and the Nuprl project [Con86]. Of these systems, only Nuprl provides a mechanism supporting a style of notational definitions close to that used conventionally. However, certain deficiencies of the Nuprl definition mechanism[1] were brought to light through great deal of practical experience with the system. This experience in turn aided in the formulation of desirable properties for such a mechanism as well as pointing out the need for a general and formal solution.

1.1 Motivation.

First, the conventional practice of introducing and using notational definitions deserves closer scrutiny. We examine both the logical nature of notational definitions and some special considerations arising from an interactive setting in which a user engages in a "dialogue with a proof system" [Mil87].

In general, assume that a formal description is given of the syntax of a language $\mathcal{L}$ that is to be extended with notational definitions. Figure 1 presents examples of notational definitions that might appear in a mathematics text, where $\mathcal{L}$ is a first-order language. (The following discussion is not meant to apply only to standard first-order languages; this language is used here only to provide a source of familiar examples.)

The abbreviative notations that we consider, such as those of Figure 1, are

[1] See Section 7 for further discussion of the Nuprl definition mechanism.

$$a \Leftrightarrow b \stackrel{\text{def}}{=} (a \Rightarrow b) \wedge (b \Rightarrow a) \quad (1)$$

$$a \mid b \stackrel{\text{def}}{=} \exists z.z > 0 \wedge a \times z = b \quad (2)$$

$$\forall x \in a.b^x \stackrel{\text{def}}{=} \forall x.(x \in a) \Rightarrow b \quad (3)$$

$$\forall x \in a, y \in b^x.c^{x,y} \stackrel{\text{def}}{=} \forall x \in a.\forall y \in b.c \quad (4)$$

$$\forall x, y \in a.b^{x,y} \stackrel{\text{def}}{=} \forall x \in a.\forall y \in a.b \quad (5)$$

$$\exists! x.a^x \stackrel{\text{def}}{=} \exists x.a \wedge (\forall y.a[y/x] \Rightarrow x = y) \quad (6)$$

Figure 1: Examples of notational definitions.

introduced by means of *equations*. These equations have the form

$$lhs \stackrel{\text{def}}{=} rhs$$

where the left-hand side, *lhs*, denotes new notation defined in terms of the right-hand side, *rhs*[2]. The *rhs* is an expression built out of expressions in $\mathcal{L}$, other instances of notational definitions, and possibly a substitution operator. Given expressions A and B the expression $A \Leftrightarrow B$ will be called an *instance* of the notation defined in Equation 1. This instance $A \Leftrightarrow B$ is understood to represent, or to stand for, the expression $(A \Rightarrow B) \wedge (B \Rightarrow A)$ which we will call the *expansion* of the instance $A \Leftrightarrow B$.

A fundamental property of notational definitions is that they must be formally nonessential [WR10,Qui53]. The use of notational definitions is justified only when all instances of defined notations can be eliminated. That is, all instances of defined notations occurring in any given expression can be expanded so that only primitive constructs in the language $\mathcal{L}$ remain. We will also require that a definition instance uniquely determine its expansion (modulo some notion of lexical equivalence) and that the result of expanding all definition instances in an expression be independent of the order these instances are expanded.

Syntactic operators that bind variables complicate the formalization of notational definitions. Such variable binding constructs appear in the languages used in all branches of mathematics and in the syntax of many programming languages. For example, in each of the familiar expressions

$$\sum_{x=1}^{y} x^2, \quad \int_0^y e^x dx, \quad \forall x \in \mathcal{N}.x \leq y, \quad \{x \in \mathcal{N} \mid x \leq y\}$$

[2]We use the more mnemonic *lhs* and *rhs* rather than *definiendum* and *definiens*.

the variable x is *bound* while the variable y is *free*. The expression $x \leq y$ is said to be in the *scope* of the binding variable x in $\forall x \in \mathcal{N}.x \leq y$. In order to address complications involving binding variables the operations of substitution, computation of free variables, and change of bound variables are identified as the *basic operations* on expressions.

Since defined notations can themselves contain binding variables, as in Equations 3—6, it should be possible to apply the basic operations directly to instances of defined notations without involving expansion. This is tantamount to requiring that defined notations have a discernible binding structure, and so implies that definitional equations must somehow specify the binding structure of defined notation. In figure 1 we have informally indicated the binding structure of the defined notations by annotating the schematic variables of the *lhs* with superscripts; the notation $v^{\vec{x}}$ is meant to indicate that v is in the scope of each variable in $\vec{x}$. In Equations 4 for example, the binding variables are represented by x and y and b is in the scope of x while c is in the scope of both x and y. The conventional style of writing such definitions does not usually include this kind of annotation and, if any care at all is paid to such matters, leaves the binding structure implicit.

Binding variables also require that expansion of defined notation be quite different from simple macro expansion. To illustrate this, the instance of the notation defined in Equation 2

$$z \mid z$$

surely should not expand to

$$\exists z.z > 0 \wedge z \times z = z$$

but rather to

$$\exists w.w > 0 \wedge z \times w = z$$

for some new variable w. We can address this problem as well as many others by requiring that expansion commute with the basic operations on expressions. This is especially important with regard to the design of interfaces to interactive proof systems. In a "dialogue with a proof system" a user may enter an expression containing instances of defined notations which is subsequently involved in a sequence of operations. Requiring that expansion commute with the basic operations insures us that, expanding any instances of defined notations in an expression before, during, or after a sequence of basic operations operations will result in an expression equivalent to the expression obtained from this sequence of operations without expansions.

Next, we consider how expansion should handle the variable names used in an instance of defined notation. Suppose that a user of an interactive proof system enters (in some unspecified way) an instance of the notation defined in Equation 4

$$\forall w \in A, u \in B.C$$

where w and u are variables and A, B, and C are expression. The user might be somewhat dismayed if expanding this definition resulted in

$$\forall z \in A.\forall v \in B[z/w].C[z, v/w, u]$$

where the binding variables w and u have been lost and replaced by variables z and v in the expansion. The user would most likely rather have the expansion result in

$$\forall w \in A.\forall u \in B.C$$

where the binding variables $\{w, u\}$ used in the expansion are those chosen occurring in the definition instance. We believe that this is more important than it might seem at first glance, especially with regard to interactive proof systems. People often attach mnemonic significance to the variables names that they choose, which in turn offers aid in negotiating large and complicated proofs. Accommodating a user's expectation regarding the preservation of variable names would help make interacting with such a system more natural.

Finally, a good framework for notational definitions should lend itself to implementations that are not overly expensive computationally. The implementation of expressions, defined notations, and expansion procedures should be reasonably efficient.

We summarize by enumerating seven properties that a framework for notational definitions should satisfy in order to be adequate for our purposes.

Property 1: Generality. We would like to proceed a formal presentation of a language $\mathcal{L}$ to a formal account of notational definitions over $\mathcal{L}$.

Property 2: All instances of defined notations can be eliminated.

Property 3: The basic operations are defined on instances of defined notations.

Property 4: Expansion must commute with the basic operations.

Property 5: Supports an equational presentation of definitions.

Property 6: Variable names are preserved on expansion.

Property 7: Lends itself to an efficient implementation.

1.2 An overview of this report.

Section 2 introduces a typed λ-calculus with operators for pairing and projection and a simple abbreviation mechanism, called a δ-equation, that allows constants to represent closed expressions. Section 3 outlines a Church-style [Chu40] method of utilizing the typed λ-calculus for language representation. Section 4 begins with an informal outline of our solution to the problem formulating an account of notational definitions that satisfies Properties 1–5 outlined above.

The body of Section 4 is devoted to a formal presentation of this solution. A new type of equation, called a Δ-equation, is introduced to model conventional definitional equations. Section 5 defines a particular expansion function that accounts for Properties 6 and 7. Section 6 discusses implementation applications of the formal account developed in Sections 2– 4. Section 7 discusses related work, including the Nuprl and EFS [Gri87] definition mechanisms. Finally, the appendix contains proofs needed to support claims made in Sections 4 and 5.

2 A typed λ-calculus.

This section presents the definition of a typed λ-calculus with pairing and a simple abbreviation mechanism that allows constants to represent closed terms.

2.1 Syntax.

We begin by defining the set of *types*. Given a non-empty finite set of *base types* T_0 define the set of $\mathcal{T}[T_0]$ of *types* over T_0 by the following grammar, where τ ranges over $\mathcal{T}[T_0]$ and t ranges over T_0:

$$\tau \ ::= \ t \ \mid \ \tau \rightarrow \tau \ \mid \ \tau \times \tau$$

We will often write the $\mathcal{T}$ for $\mathcal{T}[T_0]$ when T_0 is clear from the context.

Assume we have a denumerable set of *variables* $\mathcal{V}$ with a denumerable subset $\mathcal{V}_\tau \subset \mathcal{V}$ for each $\tau \in \mathcal{T}$. There is a denumerable set of *constants* $\mathcal{C}$ with $\mathcal{V} \cap \mathcal{C} = \phi$. The set of *expressions* is the given by the following syntax, where τ ranges over $\mathcal{T}[T_0]$, x ranges over $\mathcal{V}$, and c ranges over $\mathcal{C}$:

$$e \ ::= \ x^\tau \ \mid \ c \ \mid \ \lambda x^\tau.e \ \mid \ e(e) \ \mid \ \langle e, e \rangle \ \mid \ \pi_1(e) \ \mid \ \pi_2(e)$$

We will often write $\lambda x.e$ for $\lambda x^\tau.e$ when the type τ can be inferred from the context.

Given an ordered list of variables $[x_1, \ldots, x_n]$ we will use the notations

$$\begin{aligned} \vec{x} &\equiv [x_1, x_2, \ldots x_n] \\ \lambda\vec{x}.e &\equiv \lambda x_1.\lambda x_2. \ldots .\lambda x_n.e \\ e(\vec{x}) &\equiv e(x_1)(x_2)\ldots(x_n) \\ \{\vec{x}\} &\equiv \{x_1, x_2, \ldots x_n\} \end{aligned}$$

with the understanding that $\lambda[].e \equiv e$ and $e([]) \equiv e$.

A binary relation on expressions, $e_2 \subseteq e_1$, read e_2 is a subterm of e_1, is defined as

Definition 1 ($e_2 \subseteq e_1$) *1.* $e_1 \subseteq e_1$,

2. *for any expression* e *if* $e_2 \subseteq e_1$ *then* $e_2 \subseteq e_1(e)$, $e_2 \subseteq \langle e_1, e\rangle$, $e_2 \subseteq \langle e, e_1\rangle$, $e_2 \subseteq \lambda x.e_1$, $e_2 \subseteq e(e_1)$, *and* $e_2 \subseteq \pi_i(e_1)$.

If $e_2 \subseteq e_1$ it is possible that there is more than one occurrence of the expression e_2 contained in e_1. We will write $e_1\{e_2\}$ when we want to indicate a particular occurrence of a subterm e_2 of e_1 and $e_1\{e_2'\}$ will indicate that this occurrence of e_2 in e_1 has been replaced by e_2'.

We define the free variables of an expression in the usual way.

Definition 2 (Fvars) *For any expression* e, *define* Fvars$[e]$ *to be the set of* free variables *of* e. Fvars$[e]$ *is defined by induction on the structure of* e*:*

$$\begin{array}{ll}
\text{Fvars}[v] & \equiv \{v\} \\
\text{Fvars}[c] & \equiv \phi \\
\text{Fvars}[\lambda x.e_1] & \equiv \text{Fvars}[e_1] - \{x\} \\
\text{Fvars}[e_1(e_2)] & \equiv \text{Fvars}[e_1] \cup \text{Fvars}[e_2] \\
\text{Fvars}[\langle e_1, e_2\rangle] & \equiv \text{Fvars}[e_1] \cup \text{Fvars}[e_2] \\
\text{Fvars}[\pi_i(e_1)] & \equiv \text{Fvars}[e_1] \quad (i \in \{1,2\})
\end{array}$$

An expression e is *closed* if Fvars$[e] = \phi$. The variable x in the expression $\lambda x.b$ is said to *bind* free occurrences of x in b, and that b and all subterms of b are in the *scope* of x.

The functions Bvars and Evars are defined to obtain the set of *binding variables* and the set of *expression variables* contained in a given expression.

Definition 3 (Bvars) *For any expression* e, *define* Bvars$[e]$ *to be the set of* binding variables *of* e. Bvars$[e]$ *is defined by induction on the structure of* e*:*

$$\begin{array}{ll}
\text{Bvars}[v] & \equiv \phi \\
\text{Bvars}[c] & \equiv \phi \\
\text{Bvars}[\lambda x.e_1] & \equiv \{x\} \cup \text{Bvars}[e_1] \\
\text{Bvars}[e_1(e_2)] & \equiv \text{Bvars}[e_1] \cup \text{Bvars}[e_2] \\
\text{Bvars}[\langle e_1, e_2\rangle] & \equiv \text{Bvars}[e_1] \cup \text{Bvars}[e_2] \\
\text{Bvars}[\pi_i(e_1)] & \equiv \text{Bvars}[e_1] \quad (\textit{for } i \in \{1,2\})
\end{array}$$

Definition 4 (Evars) *For any expression* e, *define* Evars$[e]$ *to be the set of* expression variables *of* e. Evars$[e]$ *is defined by induction on the structure of* e*:*

$$\begin{array}{ll}
\text{Evars}[v] & \equiv \{v\} \\
\text{Evars}[c] & \equiv \phi \\
\text{Evars}[\lambda x.e_1] & \equiv \text{Evars}[e_1] \\
\text{Evars}[e_1(e_2)] & \equiv \text{Evars}[e_1] \cup \text{Evars}[e_2] \\
\text{Evars}[\langle e_1, e_2\rangle] & \equiv \text{Evars}[e_1] \cup \text{Evars}[e_2] \\
\text{Evars}[\pi_i(e_1)] & \equiv \text{Evars}[e_1] \quad (\textit{for } i \in \{1,2\})
\end{array}$$

Definition 5 (Vars) *For any expression* e *define the set*

$$\text{Vars}[e] \equiv \text{Evars}[e] \cup \text{Bvars}[e].$$

For example, if $e \equiv \lambda x.\lambda y.\lambda z.\langle x, w\rangle$ then

$$\begin{array}{lcllcl} \text{Fvars}[e] & \equiv & \{w\}, & \text{Bvars}[e] & \equiv & \{x, y, z\}, \\ \text{Evars}[e] & \equiv & \{x, w\}, & \text{Vars}[e] & \equiv & \{x, y, z, w\}. \end{array}$$

We use the notation

$$e\left[\begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array}\right].$$

to denote the standard notion of simultaneous capture-avoiding substitution into e of the e_i for free occurrences of the v_i (see for example Van Daalen [vD80]) and

$$e\left\{\begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array}\right\}.$$

to denote the simultaneous *replacement* of free occurrences of v_i by e_i in e, that is, substitution without capture avoidance (see for example Huet [Hue75]).

Given two expressions e_1 and e_2 we write $e_1 =_\alpha e_2$ to denote the usual notion of α-equality, i.e. the equivalence relation on expressions generated by the renaming of bound variables.

Also, assume we have a function NewVar$[S]$ that takes a finite set of variables S and returns a new variable $z \notin S$.

2.2 Typing expressions.

A *signature* Σ is a pair $(T_0, \mathcal{S})$ where T_0 is a non-empty finite set of base types and $\mathcal{S}$ is a finite list of pairwise distinct typed constants $\mathcal{S} = [c_1^{\tau_1}, \ldots, c_k^{\tau_k}]$ where $c_i \in \mathcal{C}$ and the $\tau_i \in \mathcal{T}[T_0]$.

Given $\Sigma = (T_0, \mathcal{S})$ we define the relation $\mathcal{D} \vdash_\Sigma e \in \tau$ in Figure 2.2, (also written $\mathcal{D} \vdash e \in \tau$ or $\vdash e \in \tau$ when Σ or $\mathcal{D}$ are clear from the context), to indicate that the expression e has type τ. The meta-variable $\mathcal{D}$ will range over finite lists of δ-equations[3]

$$\mathcal{D} \quad = \quad d_1 \stackrel{\delta}{=} e_1, \ldots, d_m \stackrel{\delta}{=} e_m$$

which allow for the binding of a constant to a closed expression. Here $\mathcal{D}$ is assumed to be *well-formed* in the sense that for each i,

$$d_i \notin \{d_1, \ldots, d_{i-1}\} \cup \mathcal{C},$$

and d_i does not occur in e_k for $1 \leq k \leq i$, and $\text{Fvars}[e_i] = \phi$ for each i, and that for each i there is some τ^i such that

$$d_1 \stackrel{\delta}{=} e_1, \ldots, d_{i-1} \stackrel{\delta}{=} e_{i-1} \vdash_\Sigma e_i \in \tau^i.$$

[3] These δ-equations are not those of Barendregt [Bar84] but rather are a special case of the δ-equations of the AUTOMATH (Van Daalen [vD80]) languages. AUTOMATH δ-equations allow the constant d to be parameterized. In Section 4 it will be seen that this notion corresponds to a special case of our Δ-equation.

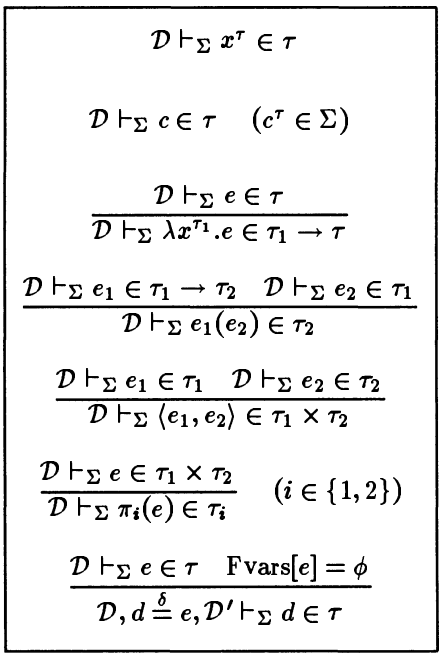

Figure 2: The typing relation.

Define the sets of expressions of a given type τ (with respect to $\mathcal{D}$) as

$$\begin{aligned}\Lambda^{\mathcal{D}}_{\tau}[\Sigma] &\equiv \{e \mid \mathcal{D} \vdash_{\Sigma} e \in \tau\},\\ \Lambda^{\mathcal{D}}_{S}[\Sigma] &\equiv \bigcup_{\tau \in S} \Lambda^{\mathcal{D}}_{\tau}[\Sigma],\\ \Lambda^{\mathcal{D}}[\Sigma] &\equiv \Lambda^{\mathcal{D}}_{T}[\Sigma].\end{aligned}$$

The set $\Lambda^{\mathcal{D}}[\Sigma]$, which we will often denote simply as Λ when $\mathcal{D}$ and Σ are clear from the context, is called the set of *well-formed* expressions.

2.3 Reduction and definitional equality.

We define a binary relation on expressions $\mathcal{R}^{\mathcal{D}}$ as

$$\mathcal{R}^{\mathcal{D}} \equiv \beta \cup \eta \cup \pi_1 \cup \pi_2 \cup \sigma \cup \delta^{\mathcal{D}}$$

where

$$\begin{aligned}\beta &= \{((\lambda x.e)(e'), e[e'/x]) \mid x \in \mathcal{V}, e_1, e_2 \in \Lambda\},\\ \eta &= \{(\lambda x.e(x), e) \mid x \in \mathcal{V}, e \in \Lambda, x \notin \mathrm{Fvars}[e]\},\\ \pi_1 &= \{(\pi_1(\langle e_1, e_2\rangle), e_1) \mid e_1, e_2 \in \Lambda\},\\ \pi_2 &= \{(\pi_2(\langle e_1, e_2\rangle), e_2) \mid e_1, e_2 \in \Lambda\},\\ \sigma &= \{\langle \pi_1(e), \pi_2(e)\rangle, e) \mid e \in \Lambda\},\\ \delta^{\mathcal{D}} &= \{(d, e) \mid d \in \mathcal{C}, e \in \Lambda, d \stackrel{\delta}{=} e \text{ in } \mathcal{D}\}.\end{aligned}$$

We will usually write $\mathcal{R}$ for $\mathcal{R}^{\mathcal{D}}$ leaving $\mathcal{D}$ to be inferred from the context.

Define the relation $e_1 \longrightarrow_{\mathcal{R}} e_2$ (in Barendregt [Bar84] this is called the compatible closure of $\mathcal{R}$) as

1. $e_1 \longrightarrow_{\mathcal{R}} e_2$ if $(e_1, e_2) \in \mathcal{R}$,
2. if $e_1 \longrightarrow_{\mathcal{R}} e_2$ then $\pi_1(e_1) \longrightarrow_{\mathcal{R}} \pi_1(e_2)$, $\pi_2(e_1) \longrightarrow_{\mathcal{R}} \pi_2(e_2)$, and $\lambda x.e_1 \longrightarrow_{\mathcal{R}} \lambda x.e_2$,
3. if $e_1 \longrightarrow_{\mathcal{R}} e_2$ then for every expression e, $e_1(e) \longrightarrow_{\mathcal{R}} e_2(e)$, $e(e_1) \longrightarrow_{\mathcal{R}} e(e_2)$, $\langle e_1, e\rangle \longrightarrow_{\mathcal{R}} \langle e_2, e\rangle$, and $\langle e, e_1\rangle \longrightarrow_{\mathcal{R}} \langle e, e_2\rangle$.

We will usually write this relation as $\longrightarrow$ leaving off the subscript.

A binary relation on expressions $\longrightarrow\!\!\!\!\!\rightarrow$ is defined as

1. $e_1 \longrightarrow\!\!\!\!\!\rightarrow e_2$ if $e_1 =_{\alpha} e_2$,
2. $e_1 \longrightarrow\!\!\!\!\!\rightarrow e_2$ if $e_1 \longrightarrow e_2$,

3. $e \longrightarrow\!\!\!\!\!\rightarrow e$,

4. if $e_1 \longrightarrow\!\!\!\!\!\rightarrow e_2$ and $e_2 \longrightarrow\!\!\!\!\!\rightarrow e_3$ then $e_1 \longrightarrow\!\!\!\!\!\rightarrow e_3$.

In words, $\longrightarrow\!\!\!\!\!\rightarrow$ is the reflexive transitive closure of the relation obtained by forming the union of the α-equality relation and the $\longrightarrow$ relation. We do this since we want to pay close attention to variable names and so do not want to identify expressions with their equivalence class modulo α-equality as is usually done.

The symbol $=_\Lambda$ will denote the equivalence relation on expressions generated by the $\longrightarrow\!\!\!\!\!\rightarrow$ relation. If $e_1 =_\Lambda e_2$ then e_1 and e_2 are said to be *definitionally equal.*

The relation $\longrightarrow$ is Church-Rosser and strongly normalizing for well-formed terms. This can be shown by a suitable mapping into a related λ-calculus, (without δ-equations), that is known to have these properties (such as the calculus of Pottinger [Pot78,Pot81], Troelstra [Tro86], or Vrijer [dV87]).

3 Representing languages à la Church.

The λ-calculus described in Section 2 will be used to represent a given object language $\mathcal{L}$. The treatment of language representation presented here is influenced most by the theory of expressions presented in the Edinburgh Logical Framework [HHP87,AHM87], from which we have adopted the notion of a signature describing a language $\mathcal{L}$, by Martin-Löf's *system of arities*, as described in Nordström, Petersson and Smith [NPS86], and by Curry and Feys [CF58]. All of these in turn have been influenced by the methods of Church [Chu40], and so we will refer to it as the Church-style of encodings languages.

A language $\mathcal{L}$ is encoded in the λ-calculus by means of a signature $\Sigma^{\mathcal{L}} \equiv (T_0^{\mathcal{L}}, S^{\mathcal{L}})$ so that each expression A of $\mathcal{L}$ can be represented as a λ-expression $\overline{A} \in \Lambda[\Sigma^{\mathcal{L}}]$. The set of base types $T_0^{\mathcal{L}}$ is comprised of one base type for each syntactic category of $\mathcal{L}$. Variables of $\mathcal{L}$ are represented by variables of the λ-calculus of the appropriate type. The set $S^{\mathcal{L}}$ is comprised of one constant c_{op} of type τ_{op} for each syntactic operator op of $\mathcal{L}$. We adopt the convention that the types of all syntactic constants are of the form $\tau \to t$ where $t \in T_0^{\mathcal{L}}$. The type τ_{op} specifies the syntactic categories of the subterms and the binding structure of the operator corresponding to the constant c_{op}. The abstraction $\lambda x.b$ is used to represent all variable binding operations of $\mathcal{L}$.

In this way the formal symbols and the formation rules of a language $\mathcal{L}$ are accounted for by typed constants and the type checking rules of the λ-calculus while notions of free and bound variables, substitution and definitional equality are inherited directly from those defined for the λ-calculus. The set of expressions in the language $\mathcal{L}$ is then identified with the set of λ-expressions $\{\overline{A} \mid A \text{ in } \mathcal{L}\}$. See [HHP87] for a discussion of the adequacy of this kind of representation.

For illustration we develop a signature $\Sigma^{fol} \equiv (T_0^{fol}, S^{fol})$ based on a small first order language with individuals, and propositions, and sets, whose gram-

			c_{op}	τ_{op}
i	::=	x		
	|	0	0	$\mathbf{i}$
	|	$i + i$	$+$	$(\mathbf{i} \times \mathbf{i}) \to \mathbf{i}$
	|	$i \times i$	$\times$	$(\mathbf{i} \times \mathbf{i}) \to \mathbf{i}$
P	::=	$i = i$	$=$	$(\mathbf{i} \times \mathbf{i}) \to \mathbf{o}$
	|	$P \wedge P$	$\wedge$	$(\mathbf{o} \times \mathbf{o}) \to \mathbf{o}$
	|	$P \vee P$	$\vee$	$(\mathbf{o} \times \mathbf{o}) \to \mathbf{o}$
	|	$P \Rightarrow P$	$\Rightarrow$	$(\mathbf{o} \times \mathbf{o}) \to \mathbf{o}$
	|	$\forall x.P$	$\forall$	$(\mathbf{i} \to \mathbf{o}) \to \mathbf{o}$
	|	$\exists x.P$	$\exists$	$(\mathbf{i} \to \mathbf{o}) \to \mathbf{o}$
	|	$i \in S$	$\in$	$(\mathbf{i} \times \mathbf{set}) \to \mathbf{o}$
S	::=	ϕ	ϕ	$\mathbf{set}$
	|	$\mathcal{N}$	$\mathcal{N}$	$\mathbf{set}$
	|	$\{x \in S \mid P\}$	Ξ	$(\mathbf{set} \times (\mathbf{i} \to \mathbf{o})) \to \mathbf{set}$

Figure 3: Grammar and syntactic constants for a first-order language.

mar is presented in together with typed constants to represent the syntactic operators, where $T_0^{fol} = \{\mathbf{i}, \mathbf{o}, \mathbf{set}\}$ represents the syntactic categories of individuals, propositions, and sets respectively. Given this encoding we have, for example,

$$\overline{A \wedge B} \equiv \wedge(\langle \overline{A}, \overline{B} \rangle),$$

$$\overline{\forall x.A} \equiv \forall(\lambda x^{\mathbf{i}}.\overline{A}),$$

for propositions A and B.

A word is in order to explain why the pairing operator has been included in the λ-calculus. A syntactic operator with n binding variables and k subexpressions can be viewed abstractly as

$$op(\vec{x}; N_1^{\vec{y_1}}, \ldots, N_k^{\vec{y_k}})$$

where $\vec{x} = \{x_1, \ldots, x_n\}$ are the binding variables and where the superscripted variables $\{\vec{y_i}\} \subseteq \{\vec{x}\}$ indicate which variables of $\vec{x}$ bind in the subexpression N_i. Such a syntactic operator can be encoded in the typed λ-calculus with a constant c_{op} of the appropriate type as

$$c_{op}(\langle \lambda \vec{y_1}.N_1, \ldots, \lambda \vec{y_k}.N_k \rangle),$$

For example, an operator

$$op(x, y; N_1, N_2^x, N_3^{x,y}, N_4^{x,y})$$

can be encoded as

$$c_{op}(\langle N_1, \lambda x.N_2, \lambda x, y.N_3, \lambda x, y.N_4\rangle).$$

However, by using the pairing operator we can represent this syntactic construct without duplicating the variables x and y as

$$c'_{op}(\langle N_1, \lambda x.\langle N_2, \lambda y.\langle N_3, N_4\rangle\rangle\rangle).$$

Of course, this approach cannot always avoid the duplication of variables, for example with an operator such as

$$op(x, y; N_1^x, N_2^y, N_3^{x,y})$$

cannot be represented without duplicating variables. Such situations seem to arise rarely and so, we believe, do not warrant abandoning the advantages of the λ-calculus as a framework for encoding languages.

Although this issue of variable duplication may not present itself in the encoding of the basic syntactic operators of a given language, such as the one presented in Figure 3, it arises very quickly when considering syntactic definitions. In the next section we will see that the ability to encode expression in this way is crucial to achieving Property 4, the equational presentation of definitions.

In the next section a formal account of notational definition is built on this Church-style representation of languages and thus inherits the generality of this representation and is able to describe notational definitions for a wide class of languages — exactly those languages that can be represented in our typed λ-calculus. These include the languages of first-order logic, higher-order logics, modal logics, and various type theories.

4 Δ-equations.

This section presents a formal account of notational definitions. We begin with an informal outline and then proceed by making the notions introduced in this discussion precise.

4.1 Informal outline.

Given the λ-representation the first-order language presented in Section 3, suppose that we want to to represent the new notation introduced in Equation 7 of Figure 4. Recall that this equation is meant to indicate the structure of the defined notation — that there are three subexpressions and two binding variables, indicated schematically by a, b, c, x, and y respectively, with a outside of the scope of x and y, b in the scope of x and outside the scope of y, and c in the scope of both x and y.

$$\forall x \in a, y \in b^x.c^{x,y} \stackrel{\text{def}}{=} \forall x \in a.\forall y \in b.c \tag{7}$$

$$\forall_2(\langle a, \lambda x.\langle b, \lambda y.c\rangle\rangle) \stackrel{\Delta}{=} \forall_1(\langle a, \lambda x.\forall_1(\langle b, \lambda y.c\rangle)\rangle) \tag{8}$$

$$\forall_2 \stackrel{\delta}{=} \lambda z^\tau.\forall_1(\langle \Phi_a[z], \lambda x.\forall_1(\langle \Phi_b[z], \lambda y.\Phi_c[z]\rangle)\rangle) \tag{9}$$

Figure 4: A conventional equation with related Δ and δ equations.

We adopt as a central thesis that a definitional equation, such as Equation 7, can be modeled by introducing a new syntactic constant into $\Lambda^{\mathcal{D}}[\Sigma^{\mathcal{L}}]$ following the same approach and conventions outlined in Section 3 for defining the basic syntactic operators of $\mathcal{L}$. Furthermore, this new constant must be introduced with a δ-equation in order to allow for its elimination.

We can, for example, treat the definitional equation given above as the declaration of a new constant $\forall_2$ of type $\tau = \mathbf{set} \times (\mathbf{i} \to \mathbf{set} \times (\mathbf{i} \to \mathbf{o}))$. In this way we extend our encoding map to account for instances of this defined notation with

$$\overline{\forall w \in A, u \in B.C} \equiv \forall_2(\langle \overline{A}, \lambda w.\langle \overline{B}, \lambda u.\overline{C}\rangle\rangle). \tag{10}$$

(Note again that the use of the pairing operator allows us to accomplish this representation without duplicating the variables w and u in the encoded instance.) This new constant can be introduced into $\mathcal{D}$ with a δ-equation

$$\forall_2 \stackrel{\delta}{=} \lambda z^\tau.\hat{e}$$

defined so that a reduction

$$\forall_2(\langle \overline{A}, \lambda w.\langle \overline{B}, \lambda u, \overline{C}\rangle\rangle) \longrightarrow (\lambda z^\tau.\hat{e})(\langle \overline{A}, \lambda w.\langle \overline{B}, \lambda u, \overline{C}\rangle\rangle) \longrightarrow\!\!\!\!\!\rightarrow\ e'$$

results in an instance e' of the λ-representation of the right-hand side of the defining equation

$$\overline{\forall x \in a.\forall y \in b.c} \equiv \forall_1(\langle a, \lambda x.\forall_1(\langle b, \lambda y.c\rangle)\rangle). \tag{11}$$

A constant $\forall_1$ of type $(\mathbf{set} \times (\mathbf{i} \to \mathbf{o})) \to \mathbf{o}$ is assumed to represent the new notation introduced in Equation 3

$$\forall x \in a.b^x \stackrel{\text{def}}{=} \forall x.(x \in a) \Rightarrow b$$

and that the encoding map is extended to account for this notation with

$$\overline{\forall w \in A.B} \equiv \forall_1(\langle \overline{A}, \lambda w.\overline{B}\rangle).$$

The expression $\hat{e}$ can be obtained by replacing each variable a, b, and c in expression 11 with expressions involving z that, loosely speaking, "take apart" the argument z on reduction while instantiating the results in the appropriate binding context. For example, replacing

$$a \quad \text{with} \quad \pi_1(z), \tag{12}$$
$$b \quad \text{with} \quad \pi_1(\pi_2(z)(x)), \tag{13}$$
$$c \quad \text{with} \quad \pi_2(\pi_2(z)(x))(y) \tag{14}$$

in expression 11 results in δ-equation 9 of Figure 4. To check that this definition of $\forall_2$ makes sense we replace z in expressions 13–14 by

$$I \equiv \langle \overline{A}, \lambda w.\langle \overline{B}, \lambda u, \overline{C}\rangle\rangle$$

to obtain the reductions

$$\begin{array}{lll} \pi_1(I) & \longrightarrow\!\!\!\!\!\rightarrow & \overline{A}, \\ \pi_1(\pi_2(I)(x)) & \longrightarrow\!\!\!\!\!\rightarrow & \overline{B}[x/w], \\ \pi_2(\pi_2(I)(x))(y) & \longrightarrow\!\!\!\!\!\rightarrow & \overline{C}[x/w][y/u], \end{array}$$

and then expand an instance of this notation by the reduction

$$\begin{array}{lll} \overline{\forall w \in A, u \in B.C} & \equiv & \forall_2(\langle \overline{A}, \lambda w.\langle \overline{B}, \lambda u.\overline{C}\rangle\rangle) \\ & \longrightarrow & (\lambda z.\hat{e})(\langle \overline{A}, \lambda w.\langle \overline{B}, \lambda u.\overline{C}\rangle\rangle) \\ & \longrightarrow\!\!\!\!\!\rightarrow & \forall_1(\langle \overline{A}, \lambda x.\forall_1(\langle \overline{B}[x/w], \lambda y.\overline{C}[x/w][y/u]\rangle)\rangle) \\ & \equiv & \overline{\forall x \in A.\forall y \in B[x/w].C[x/w][y/u].} \end{array}$$

(It is assumed, for the sake of a clear presentation, that the variables $\{x, y\}$ are not α-converted in the process of reduction.)

The main innovation presented here is the introduction of an intermediate form, called a Δ-equation, to represent definitional equations. A Δ-equation is written

$$d(\sigma) \stackrel{\Delta}{=} e, \tag{15}$$

where d is a new constant and σ, called an s-term, is either a variable, a pair of s-terms, or a λ-abstraction whose body is an s-term. The s-term σ serves to explicitly specify the binding structure of the of the defined notation as well as name its components. Such a Δ-equation is translated into a δ-equation

$$d \stackrel{\delta}{=} \mathcal{F}_e^\sigma$$

using an expression valued function $\mathcal{F}_e^\sigma$ defined below. A given expression will be called a σ-instance if it has the same type and "shape" as the s-term σ.

An expression $d(I)$ where I is a σ-instance will be called an instance of the Δ-equation given above — it represents an instance of the notation introduced by the Δ-equation. Expanding a definition instance $d(I)$ is then simply a reduction sequence starting with a δ-reduction

$$d(I) \longrightarrow \mathcal{F}_e^{\sigma}(I) \longrightarrow\!\!\!\!\!\rightarrow e'.$$

For example, representing Equation 7 with Δ-equation 8 will yield δ-equation 9 by this translation.

A Δ-equation is intended to be much closer to a conventional definitional equation than is a simple δ-equation. For example, given the encoding map discussed above, Δ-equation 8 can be viewed as

$$\overline{\forall x \in a, y \in b.c} \triangleq \overline{\forall x \in a.\forall y \in b.c}, \tag{16}$$

which is much closer to the defining equation 7 than is δ-equation 9. The relationship between conventional definitional equations and Δ-equations involves both the λ-representation of language constructs and a straightforward notational translation, and will be discussed along with other implementation concerns in Section 6.

It should be noted that the variables occurring in the left-hand side of a conventional defining equation, such as 7, are actually schematic *meta*-variables denoting locations for expression and variables, while those of the left hand side of the Δ-equation 16 are variables of Λ. It is hoped that this "pun" does not cause any confusion.

In order to provide for expansions which satisfy additional properties we define an *acceptable expansion* to be any expression valued function Exp such that $\mathrm{Exp}[d(I)]$ is definitionally equal to $d(I)$. We then define one particular function Exp^{Δ}, show it to be an acceptable expansion, and show that it satisfies Properties 6 and 7, by preserving variables on expansion in most cases, and by lending itself well to an efficient implementation.

This solution to the problem outlined in Section 1 provides a clean separation of concerns by untangling Properties 1–7 in the following way. Property 1, generality, is addressed by the use of a Church-style encoding of a given object language $\mathcal{L}$. Properties 2–4, requiring that defined notations be eliminable, that the basic operations be defined on instances of defined notations, and that expansion commute with the basic operations, are satisfied by the representation of defining equations as δ-equations and of instances of defined notations as λ-expressions. The equational presentation of new notations, Property 5, is accomplished Δ-equations, which are translated into δ-equations. Additional properties, such as Properties 6 and 7, are addressed by defining acceptable expansions satisfying these properties.

4.2 Making it precise.

We begin by defining *schematic* terms, called s-terms for short.

Definition 6 (s-terms)

1. *Any variable* v^τ *is an s-term,*
2. *If* σ_1 *and* σ_2 *are s-terms and* $\text{Vars}[\sigma_1] \cap \text{Vars}[\sigma_2] = \phi$ *then* $\langle \sigma_1, \sigma_2 \rangle$ *is an s-term,*
3. *If* σ_1 *is an s-term and* $y \notin \text{Vars}[\sigma_1]$ *then* $\lambda y.\sigma_1$ *is an s-term.*

The meta-variables $\sigma, \sigma_1, \sigma_2, \ldots$ will denote s-terms.

An expression having the "same shape" as an s-term σ is called a σ-*instance.*

Definition 7 (σ-**instances**) *Given an s-term* σ*, an expression* I *is called a* σ*-instance if one of the following holds:*

1. $\sigma \equiv v^\tau$ *and* $I \in \tau$,
2. $\sigma \equiv \langle \sigma_1, \sigma_2 \rangle$ *and* $I \equiv \langle I_1, I_2 \rangle$ *where* I_1 *is a* σ_1*-instance and* I_2 *is a* σ_2*-instance.*
3. $\sigma \equiv \lambda y^\tau.\sigma_1$ *and* $I \equiv \lambda x^\tau.I_1$ *where* I_1 *is a* σ_1*-instance.*

The meta-variables $I, I_1, I_2, \ldots$ will denote σ-instances.

For example,

$$\sigma \equiv \langle a^{\mathbf{set}}, \lambda x^{\mathbf{i}}.\langle b^{\mathbf{set}}, \lambda y^{\mathbf{i}}.c^{\mathbf{o}} \rangle \rangle \tag{17}$$

is an s-term and

$$I \equiv \langle \overline{A}, \lambda w^{\mathbf{i}}.\langle \overline{B}, \lambda u^{\mathbf{i}}.\overline{C} \rangle \rangle \tag{18}$$

is a σ instance where A and B are sets and C is a proposition in our first-order language.

Each s-term σ and each σ-instance I has a type that will be denoted as $\tau[\sigma]$.

Definition 8 ($\tau[\sigma]$) *Given an s-term* σ *and a variable* $v \in \text{Evars}[\sigma]$ *define the type* $\tau[\sigma] \in \mathcal{T}$ *by induction on the structure of* σ*:*

$$\begin{aligned} \tau[v^\tau] &\equiv \tau \\ \tau[\langle \sigma_1, \sigma_2 \rangle] &\equiv \tau[\sigma_1] \times \tau[\sigma_2] \\ \tau[\lambda y^{\tau'}.\sigma_1] &\equiv \tau' \to \tau[\sigma_1] \end{aligned}$$

It is easy to show that for all s-terms σ and all σ-instances I that $I \in \tau[\sigma]$. The s-term σ and σ-instance I given in Equations 17 and 18 are both of type $\mathbf{set} \times (\mathbf{i} \to \mathbf{set} \times (\mathbf{i} \to \mathbf{o}))$.

We next define a collection of functions, var_v^σ, sub_v^σ, scope_v^σ, and abs_v^σ that allow us to analyze a σ-instance I. (These will be important in the definition of Exp^Δ in Section 5.)

Each variable $y \in \text{Bvars}[\sigma]$ can be used to name a variable var_y^σ in I.

Definition 9 (var_y^σ) *For s-term σ and variable $y \in \mathrm{Bvars}[\sigma]$ define the variable $\mathrm{var}_y^\sigma[I]$ by induction on the structure of σ:*

$$\mathrm{var}_y^{\langle\sigma_1,\sigma_2\rangle}[\langle I_1, I_2\rangle] \equiv \mathrm{var}_y^{\sigma_i}[I_i] \quad (\textit{for } y \in \mathrm{Bvars}[\sigma_i] \textit{ where } i \in \{1,2\})$$

$$\mathrm{var}_y^{\lambda z.\sigma_1}[\lambda x.I_1] \equiv \mathrm{var}_y^{\sigma_1}[I_1] \quad (z \neq y)$$

$$\mathrm{var}_y^{\lambda y.\sigma_1}[\lambda x.I_1] \equiv x$$

We can think of each variable $v \in \mathrm{Evars}[\sigma]$ as naming the subterm $\mathrm{sub}_v^\sigma[I]$ of I.

Definition 10 (sub_v^σ) *For all s-terms σ', σ-instances I, and $v \in \mathrm{Evars}[\sigma]$, define the expression $\mathrm{sub}_v^\sigma[I]$ by induction on the structure of σ:*

$$\begin{aligned}
\mathrm{sub}_v^v[I] &\equiv I \\
\mathrm{sub}_v^{\langle\sigma_1,\sigma_2\rangle}[\langle I_1, I_2\rangle] &\equiv \mathrm{sub}_v^{\sigma_i}[I_i] \quad (v \in \mathrm{Evars}[\sigma_i]) \\
\mathrm{sub}_v^{\lambda y.\sigma_1}[\lambda x.I_1] &\equiv \mathrm{sub}_v^{\sigma_1}[I_1]
\end{aligned}$$

The function $\mathrm{scope}_v^\sigma[I]$ computes the binding environment at of the subterm $\mathrm{sub}_v^\sigma[I]$ in I.

Definition 11 (scope_v^σ) *For all s-terms σ, variables $v \in \mathrm{Evars}[\sigma]$, and σ-instances I, define the vector of variables $\mathrm{scope}_v^\sigma[I]$ by induction on the structure of σ:*

$$\begin{aligned}
\mathrm{scope}_v^v[I] &\equiv [] \\
\mathrm{scope}_v^{\langle\sigma_1,\sigma_2\rangle}[\langle I_1, I_2\rangle] &\equiv \mathrm{scope}_v^{\sigma_i}[I_i] \\
&\quad (\textit{for } v \in \mathrm{Evars}[\sigma_i] \textit{ where } i = 1 \textit{ or } i = 2\ ) \\
\mathrm{scope}_v^{\lambda y.\sigma_1}[\lambda x.I_1] &\equiv x, \mathrm{scope}_v^{\sigma_1}[I_1]
\end{aligned}$$

Since every s-term σ is itself a σ-instance, the expression $\mathrm{scope}_v^\sigma[\sigma]$ is well defined for any $v \in \mathrm{Evars}[\sigma]$.

These definitions can be illustrated with the s-term σ and the σ-instance I given in Equations 17 and 18:

$$\begin{aligned}
\mathrm{var}_x^\sigma[I] &\equiv w, & \mathrm{sub}_a^\sigma[I] &\equiv \overline{A}, \\
\mathrm{var}_y^\sigma[I] &\equiv u, & \mathrm{sub}_b^\sigma[I] &\equiv \overline{B}, \\
& & \mathrm{sub}_c^\sigma[I] &\equiv \overline{C}, \\
\mathrm{scope}_a^\sigma[\sigma] &\equiv [], & \mathrm{scope}_a^\sigma[I] &\equiv [], \\
\mathrm{scope}_b^\sigma[\sigma] &\equiv [x], & \mathrm{scope}_b^\sigma[I] &\equiv [w], \\
\mathrm{scope}_c^\sigma[\sigma] &\equiv [x, y], & \mathrm{scope}_c^\sigma[I] &\equiv [w, u].
\end{aligned}$$

We now define an expression $\Phi_v^\sigma[e]$ so that for any σ-instance I we can obtain by reduction of $\Phi_v^\sigma[z][I/z]$ the expression $\mathrm{sub}_v^\sigma[I]$ with variables of $\mathrm{scope}_v^\sigma[I]$ replaced by those of $\mathrm{scope}_v^\sigma[\sigma]$.

Definition 12 (Φ_v^σ) *Given an s-term σ and a variable $v \in \mathrm{Evars}[\sigma]$ and any expression $e \in \tau[\sigma]$ define the expression $\Phi_v^\sigma[e]$ by induction on the structure of σ:*

$$\begin{array}{lll} \Phi_v^v[e] & \equiv & e \\ \Phi_v^{\langle\sigma_1,\sigma_2\rangle}[e] & \equiv & \Phi_v^{\sigma_i}[\pi_i(e)] \quad (v \in \mathrm{Evars}[\sigma_i],\ i=1 \text{ or } i=2) \\ \Phi_v^{\lambda y.\sigma_1}[e] & \equiv & \Phi_v^{\sigma_1}[e(y)] \end{array}$$

From the s-term σ defined in Equation 17 we have

$$\begin{aligned} \Phi_a^\sigma &\equiv \pi_1(z), \\ \Phi_b^\sigma &\equiv \pi_1(\pi_2(z)(x)), \\ \Phi_c^\sigma &\equiv \pi_2(\pi_2(z)(x))(y). \end{aligned}$$

Note that these are exactly expressions 13–14 from above.

Definition 13 (abs_v^σ) *For all s-terms σ, for all variables $v \in \mathrm{Evars}[\sigma]$, and for all σ-instances I, define*

$$\mathrm{abs}_v^\sigma[I] \equiv \lambda \mathrm{scope}_v^\sigma[I].\mathrm{sub}_v^\sigma[I].$$

Definition 14 (f_v^σ) *Given an s-term σ and a variable $v \in \mathrm{Evars}[\sigma]$ define the expression*

$$f_v^\sigma \equiv \lambda z^{\tau[\sigma]}.\lambda \mathrm{scope}_v^\sigma[\sigma].\Phi_v^\sigma[z]$$

where z is a new variable such that $z \notin \mathrm{Vars}[\sigma]$.

The following Lemma tells us that we can obtain (modulo α-conversion) $\mathrm{abs}_v^\sigma[I]$ by reduction of the expression $f_v^\sigma(I)$.

Lemma 1 *If σ is an s-term with $v \in \mathrm{Evars}[\sigma]$ and I is a σ-instance then*

$$f_v^\sigma(I) \twoheadrightarrow \mathrm{abs}_v^\sigma[I].$$

Proof. *See 8 in the appendix.* □

We can illustrate this using σ and I of Equations 17 and 18

$$\begin{array}{lll} \mathrm{abs}_a^\sigma[I] \equiv \overline{A}, & & f_a^\sigma(I) \twoheadrightarrow \overline{A}, \\ \mathrm{abs}_b^\sigma[I] \equiv \lambda w.\overline{B}, & & f_b^\sigma(I) \twoheadrightarrow \lambda x.\overline{B}[x/w], \\ \mathrm{abs}_c^\sigma[I] \equiv \lambda w.\lambda u.\overline{C} & & f_c^\sigma(I) \twoheadrightarrow \lambda x.\lambda y.\overline{C}[x/w][y/u]. \end{array}$$

Definition 15 ($\mathcal{F}_e^\sigma$) *For any s-term* σ *with* $\text{Evars}[\sigma] = \{v_1 \ldots v_n\}$ *and any expression* e *define*

$$\mathcal{F}_e^\sigma \equiv \lambda z^{\tau[\sigma]}.e\left\{ \frac{\Phi_{v_1}^\sigma[z]}{v_1} \quad \ldots \quad \frac{\Phi_{v_n}^\sigma[z]}{v_n} \right\}.$$

for $z = \text{NewVar}[\text{Vars}[\sigma] \cup \text{Vars}[e]]$.

In order to insure that $\mathcal{F}_e^\sigma$ is well-formed we will require that e be *compatible with* σ.

Definition 16 (e **is compatible with** σ) *An expression* $e \in \Lambda$ *is* compatible *with an s-term* σ *if*

1. $\text{Fvars}[\sigma]$ $(= \text{Evars}[\sigma])$ $= \text{Fvars}[e]$,
2. $\text{Bvars}[\sigma] \subseteq \text{Bvars}[e]$,
3. $\text{Evars}[\sigma] \cap \text{Bvars}[e] = \phi$,
4. *if* $v \in \text{Evars}[\sigma]$ *is in the scope of* $y \in \text{Bvars}[\sigma]$ *in* σ *then every occurrence of* v *in* e *is in the scope of some occurrence of* y *in* e,
5. *the type associated with each variable in* $\text{Vars}[\sigma]$ *matches the type associated with the corresponding variable in* e.

Lemma 2 *If* e *is compatible with* σ *then*

1. $\text{Fvars}[\mathcal{F}_e^\sigma] = \phi$.
2. $\mathcal{F}_e^\sigma$ *is well formed.*

We can now define Δ-equations:

Definition 17 (Δ**-equations**) *Given an s-term* σ *and an expression* e *compatible with* σ *and a new constant* d

1. *The equation*
$$d(\sigma) \triangleq e$$
is called a Δ*-equation, or the defining* Δ*-equation for* d.
2. *The meaning of such a* Δ*-equation is given by the* δ*-equation*
$$d \stackrel{\delta}{=} \mathcal{F}_e^\sigma$$
which is a well formed δ*-equation by Lemma 2.*
3. *If* I *is a* σ*-instance the expression* $d(I)$ *is called an instance of the notation introduced by the defining* Δ*-equation for* d.

As a trivial example, it is easy to check that for any s-term σ

$$d(\sigma) \triangleq \sigma \tag{19}$$

is a well-formed Δ-equation. In addition, we have

Lemma 3 *If σ is an s-term and I is an σ-instance then*

$$\mathcal{F}^{\sigma}_{\sigma}(I) \longrightarrow\!\!\!\!\!\rightarrow I.$$

This is exactly what we should expect from Equation 19.

Returning to the example given above, from Δ-equation 8 of Figure 4 the expression $\mathcal{F}^{\sigma}_{e}$ yields δ-equation 9 so that

$$\overline{\forall w \in A, u \in B.C} \longrightarrow\!\!\!\!\!\rightarrow \overline{\forall x \in A. \forall y \in B[x/w].C[x/w][y/u].}$$

One interesting special case concerns Δ-equations with $\mathrm{Bvars}[\sigma] = \phi$. In this case we can assume, without loss of generality, that

$$\sigma \equiv \langle v_1, \langle v_2, \langle \ldots, v_n \rangle \ldots \rangle.$$

A σ-instance then has the form

$$I \equiv \langle e_1, \langle e_2, \langle \ldots, e_n \rangle \ldots \rangle.$$

and

$$\Phi^{\sigma}_{v_i}[I] \longrightarrow\!\!\!\!\!\rightarrow e_i$$

so that

$$\mathcal{F}^{\sigma}_{e}(I) \longrightarrow\!\!\!\!\!\rightarrow e' =_{\Lambda} e\left[\begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array}\right].$$

This case corresponds to the AUTOMATH [vD80] δ-equation that allows the defined constant to take parameters

$$d(v_1, \ldots, v_n) \stackrel{\delta}{=} e \tag{20}$$

where the v_i are pairwise distinct variables, and where the expansion is defined as

$$d(e_1, \ldots, e_n) \longrightarrow e\left[\begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array}\right].$$

It should be noted here why parameterized δ-equations such as 20 are not adequate for our purposes. Δ-equation 7 can be modeled with a parameterized δ-equation not involving the pairing and projection operators as

$$d(f_1, \ldots, f_n) \stackrel{\delta}{=} e\left\{\begin{array}{ccc} e'_1 & \cdots & e'_n \\ \overline{v_1} & & \overline{v_n} \end{array}\right\},$$

where $\mathrm{Evars}[\sigma] = \{v_1, \ldots, v_n\}$, $e'_i = f_i(\mathrm{scope}^{\sigma}_{v_i}[\sigma])$. Note that an instance $d(I)$ of Δ-equation could then be represented as

$$d(abs^{\sigma}_{v_1}[I], \ldots, abs^{\sigma}_{v_1}[I])$$

which may involve the duplication of variables.

Figure 5 displays Equations 1-5 of Figure 1 along with corresponding Δ-equations.

$a \Leftrightarrow b$	$\stackrel{def}{=}$	$(a \Rightarrow b) \wedge (b \Rightarrow a)$
$\Leftrightarrow(\langle a, b\rangle)$	$\triangleq$	$\wedge(\langle \Rightarrow(\langle a, b\rangle), \Rightarrow(\langle b, a\rangle), \rangle)$
$a \mid b$	$\stackrel{def}{=}$	$\exists z.z > 0 \wedge a \times z = b$
$\mid(\langle a, b\rangle)$	$\triangleq$	$\exists(\lambda z. \wedge(\langle >(\langle z, 0\rangle), =(\langle \times(\langle a, z\rangle), b\rangle)))$
$\forall x \in a.b^x$	$\stackrel{def}{=}$	$\forall x.(x \in a) \Rightarrow b$
$\forall_1(\langle a, \lambda x.b)$	$\triangleq$	$\forall(\lambda x. \Rightarrow(\langle \in(\langle x, a\rangle), b\rangle)$
$\forall x \in a, y \in b^x.c^{x,y}$	$\stackrel{def}{=}$	$\forall x \in a.\forall y \in b.c$
$\forall_2(\langle a, \lambda x.\langle b, \lambda y.c\rangle\rangle)$	$\triangleq$	$\forall_1(\langle a, \lambda x.\forall_1(\langle b, \lambda y.c\rangle)\rangle)$
$\forall x, y \in a.b^{x,y}$	$\stackrel{def}{=}$	$\forall x \in a.\forall y \in a.b$
$\forall'_2(\langle a, \lambda x.\lambda y.b\rangle)$	$\triangleq$	$\forall_1(\langle a, \lambda x.\forall_1(\langle a, \lambda y.b\rangle)\rangle)$
$\exists!x.a^x$	$\stackrel{def}{=}$	$\exists x.a \wedge (\forall y.a[y/x] \Rightarrow x = y)$
$\exists!(\lambda x.a)$	$\triangleq$	$\exists(\lambda x. \wedge(\langle a, \forall(\lambda y. \Rightarrow(\langle(\lambda x.a)(y), =(\langle x, y\rangle)\rangle)))))$

Figure 5: Defining equations with corresponding Δ-equations.

4.3 Acceptable expansions.

It may be desirable to expand instance $d(I)$ of defined notations in some way other than the reduction of $\mathcal{F}_e^\sigma(I)$, and so we define the notion of an *acceptable expansion.* This will allow us to replace any occurrence of the instance $d(I)$ in an expression $e_1\{d(I)\}$ with $\mathrm{Exp}[d(I)]$ to obtain $e_1\{\mathrm{Exp}[d(I)]\}$, provided that Exp is an acceptable expansion.

Definition 18 (Acceptable Expansion)
An expression valued function $\mathrm{Exp}_\mathcal{D}[d(I)]$ *is called an acceptable expansion if for every* Δ*-equation*

$$d(\sigma) \triangleq e$$

and every σ*-instance* I

$$d(I) =_\Lambda \mathrm{Exp}_\mathcal{D}[d(I)].$$

We will usually write $\mathrm{Exp}_\mathcal{D}[d(I)]$ as $\mathrm{Exp}[d(I)]$ leaving the set of δ-equations $\mathcal{D}$ implicit.

The definition of $\mathcal{F}_e^\sigma$ can now be viewed as providing a correctness criterion for acceptable expansions. Also, it is easy to check that any acceptable expansion commutes with the basic operations on terms, and so satisfies Property 4.

Lemma 4 *If* Exp *is an acceptable expansion then for every* Δ*-equation*

$$d(\sigma) \triangleq e$$

and every σ*-instance* I

1. $\mathrm{Fvars}[d(I)] = \mathrm{Fvars}[\mathrm{Exp}[d(I)]]$
2. $\mathrm{Exp}[d(I)[e_1/v_1, \ldots, e_n/v_n]] =_\Lambda \mathrm{Exp}[d(I)][e_1/v_1, \ldots, e_n/v_n]$
3. *if* $I =_\Lambda I'$ *then* $\mathrm{Exp}[d(I)] =_\Lambda \mathrm{Exp}[d(I')]$

5 The expansion Exp^Δ.

Expanding $\overline{\forall w \in A, u \in B.C}$ in the example given above resulted in the loss of the binding variables $\{w, u\}$. In this section we define one particular acceptable expansion Exp^Δ, which provides for the preservation of variable names (Property 6) in most cases. We will also see that Exp^Δ lends itself to a relatively efficient implementation (Property 7).

5.1 Generalized substitution.

We now introduce a new definition of simultaneous substitution. Given an expression e, expressions $e_1,\ldots,e_n$, sets of variables $S_1,\ldots,S_n$ with $S_i \subseteq \mathrm{Fvars}[e_i]$, and pairwise distinct variables $v_1,\ldots,v_n$, the notation

$$e\left[\begin{array}{ccc} \frac{(e_1,S_1)}{v_1} & \cdots & \frac{(e_n,S_n)}{v_n} \end{array}\right]$$

will denote the simultaneous substitution of e_i for all free occurrences of v_i in e, allowing the possibility of capture in e_i of variables in S_i.

Simultaneous substitution and simultaneous replacement correspond to special cases of this generalized substitution. The standard notion of simultaneous capture-avoiding substitution

$$e\left[\begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array}\right].$$

can be viewed as representing the special case where $S_i = \phi$ $(i = 1,\ldots,n)$. Simultaneous replacement of free occurrences of v_i by e_i in e.

$$e\left\{\begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array}\right\}.$$

can be viewed as representing the special case where $S_i = \mathrm{Fvars}[e_i]$.

The notation

$$e\left[\begin{array}{ccc} \frac{(e_1,S_1)}{v_1} & \cdots & \frac{(e_n,S_n)}{v_n} \end{array}\right]/\{v_j\}$$

denotes substitution with the j^{th} component of the domain deleted. Also, assume we have a function NewVar[S] that takes a finite set of variables S and returns a new variable $z \notin S$.

Definition 19 (substitution) *Let $v_1,\ldots v_n$ be n $(n > 0)$ pairwise distinct variables. Let $e_1,\ldots e_n$ be expressions and $S_1,\ldots S_n$ be sets of variables such that $S_i \subseteq \mathrm{Fvars}[e_i]$. Let*

$$e^* \;=\; e\left[\begin{array}{ccc} \frac{(e_1,S_1)}{v_1} & \cdots & \frac{(e_n,S_n)}{v_n} \end{array}\right]$$

be an abbreviation for the simultaneous capture-incurring substitution of e_i for v_1 in e that is defined by induction on the structure of e and the integer n as follows:

$$\begin{array}{lll} c^* & \equiv c & \\ v^* & \equiv v & (\textit{if } v \notin \{v_1 \ldots v_n\}) \\ & \equiv e_i & (\textit{if } v \equiv v_i) \end{array}$$

$\langle e', e'' \rangle^* \quad \equiv \langle e'^*, e''^* \rangle$

$(\pi_i(e'))^* \quad \equiv \pi_i(e'^*) \quad (i \in \{1,2\})$

$(e'(e''))^* \quad \equiv e'^*(e''^*)$

$(\lambda x.e')^*$

Case 1: $n = 1$ *and* $x \equiv v_1$

$\equiv \lambda x.e'$

Case 2: $n > 1$ *and* $x \equiv v_j$

$$\equiv (\lambda x.e') \left[\frac{(e_1, S_1)}{v_1} \quad \ldots \quad \frac{(e_n, S_n)}{v_n} \right] / \{v_j\}$$

Case 3: $x \notin \vec{v}$ *and* $x \notin \bigcup_{1 \leq i \leq n} (\mathrm{Fvars}[e_i] - S_i)$

$\equiv \lambda x.e'^*$

Case 4: $x \notin \vec{v}$ *and* $x \in \bigcup_{1 \leq i \leq n} (\mathrm{Fvars}[e_i] - S_i)$.

$$\equiv \lambda z.(e' \left[\frac{z}{x} \right])^*$$

where $z = \mathrm{NewVar}[\mathrm{Vars}[e'] \cup \{\vec{v}\} \cup (\bigcup_{1 \leq i \leq n} (\mathrm{Fvars}[e_i] - S_i))]$

5.2 Simultaneous α-conversion.

In order to address the preservation of variable names we must pay close attention to variable names. To this end, we define simultaneous α-conversion.

Definition 20 (simultaneous α-conversion)
Let e be any expression. Given pairwise distinct variables

$$\vec{v} \equiv [v_1, v_2, \ldots v_n]$$

and variables (not necessarily distinct)

$$\vec{w} \equiv [w_1, w_2, \ldots w_n]$$

with $\{\vec{v}\} \cap \{\vec{w}\} = \phi$ *and* $\mathrm{Fvars}[e] \cap \{\vec{w}\} = \phi$ *define the simultaneous alpha-replacement of* v_i *by* w_i *in e, written*

$$e[\vec{w}/\vec{v}]^\alpha,$$

by induction on the structure of e:

$$
\begin{aligned}
c[\vec{w}/\vec{v}]^{\alpha} &\equiv c \\
v[\vec{w}/\vec{v}]^{\alpha} &\equiv v \\
\langle e_1, e_2\rangle[\vec{w}/\vec{v}]^{\alpha} &\equiv \langle e_1[\vec{w}/\vec{v}]^{\alpha}, e_2[\vec{w}/\vec{v}]^{\alpha}\rangle \\
(\pi_i(e_1))[\vec{w}/\vec{v}]^{\alpha} &\equiv \pi_i(e_1[\vec{w}/\vec{v}]^{\alpha}) \quad (i \in \{1,2\}) \\
(e_1(e_2))[\vec{w}/\vec{v}]^{\alpha} &\equiv e_1[\vec{w}/\vec{v}]^{\alpha}(e_2[\vec{w}/\vec{v}]^{\alpha}) \\
(\lambda x.e_1)[\vec{w}/\vec{v}]^{\alpha} & \\
&\quad \textit{Case 1: } x \notin \{\vec{v}\} \cup \{\vec{w}\} \\
&\quad \equiv \lambda x.e_1[\vec{w}/\vec{v}]^{\alpha} \\
&\quad \textit{Case 2: } x \equiv v_j \\
&\quad \equiv (\lambda w_j.e_1[\vec{w}/\vec{v}]^{\alpha}[w_j/v_j]) \\
&\quad \textit{Case 3: } x \equiv w_j \\
&\quad \equiv (\lambda z.e_1[z/x][\vec{w}/\vec{v}]^{\alpha}) \\
&\qquad \textit{where } z = \mathrm{NewVar}[\mathrm{Vars}[e_1] \cup \{\vec{v}\} \cup \{\vec{w}\}]
\end{aligned}
$$

Notice that in the second case of the last clause the condition $\mathrm{Fvars}[e_1] \cap \{\vec{w}\} = \phi$ holds since by assumption $\mathrm{Fvars}[\lambda x.e_1] \cap \{\vec{w}\} = \phi$ so $\mathrm{Fvars}[e_1] \subseteq (\mathrm{Fvars}[e_1] \cup \{x\})$ but in this case $x \notin \{\vec{w}\}$. Lemma 11 of the appendix proves that $e[\vec{w}/\vec{v}]^{\alpha} =_{\alpha} e$.

So for example, if $\vec{v} \equiv [x, y]$ and $\vec{w} \equiv [u, t]$ then

$$(\lambda u.\lambda x.\lambda y.c(u)(x)(y))[\vec{w}/\vec{v}]^{\alpha} \equiv \lambda z.\lambda u.\lambda t.c(z)(u)(t)$$

where z is a new variable.

5.3 Exp^{Δ}.

Next, assume that the set of variables $\mathcal{V}$ is partitioned into two sets,

$$\mathcal{V} = \mathcal{V}_s \cup \mathcal{V}_r$$

with

$$\mathcal{V}_s \cap \mathcal{V}_r = \phi.$$

The set $\mathcal{V}_s$ is called the set of *schematic*-variables, while the set $\mathcal{V}_r$ is called the set of *regular*-variables. Also assume that for every s-term σ that $\mathrm{Vars}[\sigma] \subseteq \mathcal{V}_s$ and that for every Δ-equation

$$d(\sigma) \stackrel{\Delta}{=} e$$

we have $(\text{Vars}[e] - \text{Vars}[\sigma]) \subseteq \mathcal{V}_r$. Further, assume for expressions e' (not the right-hand side of a Δ-equation) that $\text{Vars}[e'] \subseteq \mathcal{V}_r$.

Definition 21 *For an s-term σ, a σ-instance I is said to have a variable* free and bound w.r.t. σ *if*

$$\text{Fvars}[I] \cap \{\text{var}_y^\sigma[I] \mid y \in \text{Bvars}[\sigma]\} \neq \phi.$$

Finally, assume that we have a function $\text{ConvInst}^\sigma[I]$ such that for any σ-instance I

1. $I =_\alpha \text{ConvInst}^\sigma[I]$,
2. $\text{ConvInst}^\sigma[I]$ is a σ-instance,
3. $\text{ConvInst}^\sigma[I]$ has no variables free and bound w.r.t. σ,
4. if I has no variables free and bound w.r.t. σ then $\text{ConvInst}^\sigma[I] \equiv I$.

In other words, $\text{ConvInst}^\sigma[I]$ α-converts bound variables in I that are free and bound w.r.t. σ to variables that are not.

Definition 22 (Exp^Δ) *Given a Δ-equation*

$$d(\sigma) \triangleq e$$

and a σ-instance I, let

$$\begin{array}{lcl} \vec{y} & = & [y_1, \ldots, y_k], \\ I' & = & \text{ConvInst}^\sigma[I], \\ \vec{u} & = & [\text{var}_{y_1}^\sigma[I'], \ldots, \text{var}_{y_k}^\sigma[I']], \\ e_i & = & \text{sub}_{v_i}^\sigma[I'], \\ \vec{w_i} & = & \text{scope}_{v_i}^\sigma[I']. \end{array}$$

Define $\text{Exp}^\Delta[d(I)]$ *as*

$$\text{Exp}^\Delta[d(I)] \equiv \tilde{e}$$

where

$$\tilde{e} \equiv e \left[\begin{array}{c} \vec{u} \\ \vec{y} \end{array} \right]^\alpha \left[\begin{array}{ccc} \dfrac{(\text{sub}_{v_1}^\sigma[I'], \{\text{scope}_{v_1}^\sigma[I']\})}{v_1} & \cdots & \dfrac{(\text{sub}_{v_n}^\sigma[I'], \{\text{scope}_{v_n}^\sigma[I']\})}{v_n} \end{array} \right].$$

Informally, the function $\text{Exp}^\Delta[d(I)]$ first changes the variables of e to correspond to those actually used an instance I and then inserts each subterm of I into the appropriate location in the result.

Returning again to our running example we have

$\text{Exp}^\Delta[\overline{\forall w \in A, u \in B.C}]$

$$\equiv \overline{\forall x \in a.\forall y \in b.c}\left[\begin{array}{c}\frac{w}{x},\frac{u}{y}\end{array}\right]^{\alpha}\left[\begin{array}{c}\frac{(\overline{A},\phi)}{a},\frac{(\overline{B},\{w\})}{b},\frac{(\overline{C},\{w,u\})}{c}\end{array}\right]$$

$$\equiv \overline{\forall w \in a.\forall u \in b.c}\left[\begin{array}{c}\frac{(\overline{A},\phi)}{a},\frac{(\overline{B},\{w\})}{b},\frac{(\overline{C},\{w,u\})}{c}\end{array}\right]$$

$$\equiv \forall_1(\langle \overline{A}, \lambda w.\forall_1(\langle \overline{B}, \lambda u.\overline{C}\rangle)\rangle)$$

$$\equiv \overline{\forall w \in A.\forall u \in B.C}$$

assuming for sake of a clear presentation that ConvInst$[I] \equiv I$, that is

$$\text{Fvars}[\langle \overline{A}, \lambda w.\forall_1(\langle \overline{B}, \lambda u.\overline{C}\rangle)\rangle] \cap \{w,u\} = \phi.$$

A few remarks concerning the function Exp^{Δ} are in order. First, we have

Theorem 5 Exp^{Δ} *is an acceptable expansion.*

Proof. See Theorem 14 in the appendix. □

Second, the expansion $\text{Exp}^{\Delta}[d(I)]$ provides for the preservation of variable names in most instances. It is only when a binding variable is free and bound w.r.t. σ in I that it is renamed in the expansion $\text{Exp}^{\Delta}[d(I)]$. This situation seems to arise rarely in practice.

Third, the expansion $\text{Exp}^{\Delta}[d(I)]$ lends itself well to an efficient implementation (Property 7) in two respects.

1. There is no need to actually construct the expression $\mathcal{F}_e^{\sigma}$ to obtain this expansion.
2. If the types of all variables in Evars$[\sigma]$ are in T_0 and if e and I are both in normal form, then $\text{Exp}^{\Delta}[d(I)]$ is obtained without reduction and is itself in normal form.

Fourth, there is no danger of violating the assumptions concerning the partition of $\mathcal{V}$ since it is easy to see that $\text{Vars}[\text{Exp}^{\Delta}[d(I)]] \subseteq \mathcal{V}_r$.

Finally, if Bvars$[\sigma] = \phi$, corresponding to an AUTOMATH-like parameterized δ equation, the definition of Exp^{Δ} simplifies to

$$\text{Exp}^{\Delta}[\langle e_1,\ldots,e_n\rangle\ldots)] \equiv e\left[\begin{array}{ccc}\frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n}\end{array}\right],$$

which is exactly the expansion for AUTOMATHs parameterized δ-equations.

6 Implementation considerations.

6.1 Implementing $\mathcal{L}$.

In order to implement a language $\mathcal{L}$ together with a facility for notational definitions it is not necessary to implement all of $\Lambda[\Sigma^{\mathcal{L}}]$, although this certainly

is an option. We could, for example, implement only a subset $\theta \subseteq \Lambda[\Sigma^{\mathcal{L}}]$, and in particular, $\Theta = \{\overline{A} \mid A \text{ in } \mathcal{L}\}$. The framework presented above can be used as a criterion to judge the correctness of a definition mechanism for such an implementation. For example, suppose that rep$[e]$ is a mapping from θ to some machine representation of expressions. Then a procedure Ψ operating on this implementation is an acceptable expansion if and only if

$$Exp^{\Psi}[d(I)] \equiv \text{rep}^{-1}[\Psi[\text{rep}[d(I)]]]$$

is an acceptable expansion.

6.2 Declaration of defined notation.

We will discuss here one possible input style for definitional equations, called the Nuprl-style, after the mechanism of that system. A declaration of new notation has the form

Define name
$p_1\ s_1\ p_2, \ldots, p_n\ s_n\ p_{n+1}$ == *rhs*

where *name* is a name associated with the equation, the p_i denote print instructions, and the s_j denote the locations of binding variables or subterms. In Nuprl the s_i are of the form $\langle u_i \rangle$ where u_i is a variable and the brackets are used to distinguish such variables from the surrounding print instructions. This notation could be extended so that the s_i contain annotations of scoping information. In either case, an algorithm is required to construct, if possible, a σ compatible with $\overline{rhs}$. If such a σ can be constructed the equation is then interpreted to represent the Δ-equation

$$name(\sigma) \triangleq \overline{rhs}.$$

The Nuprl-style of declaration has the advantage of combining the declaration of new notation with the declaration of the display of this notation.

Note that the assumptions regarding the partition of variables used in defining the expansion Exp^{Δ} do not cause any problem in implementing an expansion based on Exp^{Δ}. Variables of the *rhs* of a defining equation could either be of a special lexical class, or internally altered to be so.

7 Related work.

7.1 Lisp macros.

Lisp macros are not abbreviative in nature but are essentially very general program-valued programs. A preprocessor executes these macro programs to eliminate all macros before compilation. Macro elimination may involve the

unintended capture of free variables of arguments to a macro, and so poses a problem somewhat similar to the one addressed above for notational definitions. This problem has been addressed by [KFFD87] with their definition of "hygienic macro expansion" for Lisp.

7.2 Nuprl.

The Nuprl system provides a very flexible declaration style for definitions (see Section 6). Each Nuprl term has associated with it a display form which instructs the system on how to display the term to a user. The definition mechanism allows a user to define new display forms for terms. Instances of defined notations in Nuprl terms are always fully expanded internally, with the user seeing only the defined display forms. There seem to be three main drawbacks to this approach. First, a user cannot be certain about the form of a fully expanded term since variables of the defined notation or of the fully expanded term can capture, or be captured by, hidden variables. A user might think that some variable in an expression is free only to discover that is is captured by a hidden variable in the fully expanded term. Second, substitution may not be able to recover a defined display form when α-conversion takes place on the binding variables of an instance of defined notation. As a result, substitution into instances of defined notations during proof construction can induce a cascade of unintended definition expansions, much to the dismay of a user. Third, reestablishing display forms after substitution into instances of defined notations is computationally expensive.

These problems could be overcome by designing a mechanism based on a syntactic specification Σ^{Nuprl} for the syntax of the Nuprl language.

7.3 EFS.

The framework developed in this paper was used by the author in the implementation of a definition mechanism in the EFS [Gri87][4], an **E**nvironment for **F**ormal **S**ystems, that supports the interactive definition of formal systems and the construction of proofs in systems so defined. This system supports both the Theory of constructions [CH85] and the Edinburgh Logical Framework [HHP87]. These languages share a common syntax as typed λ-calculi with dependent types where terms and types are of the same syntactic category, and can both be represented, using the approach outlined in Section 3, with the same syntactic signature $\Sigma^{\lambda\Pi}$, having only one base type **o**.

The EFS implements only those expressions of type **o** in normal form, as well as an expansion procedure based on Exp^{Δ}. It supports a Nuprl-style of

[4] We are referring here to EFS version 2. EFS version 1, which is available in Release 2.0 of the Synthesizer Generator, and is documented in [Gri87], was implemented before the analysis of this paper was complete. In other words, beware of bugs.

declaration. A structure-oriented editor allows defined notations to be entered as templates.

There is a potential source of confusion that should be addressed. Both the Theory of Constructions and the Edinburgh Logical Framework can be used to encode the syntax of a language $\mathcal{L}$ in a manner similar to that developed in Section 3. The EFS allows a user to encode such a language interactively, and once defined, to use the definition mechanism to introduce new notations for this language. It should be noted that these definitions are taking place at the level of the representation $\Sigma^{\lambda\Pi}$, and not some representation $\Sigma^{\mathcal{L}}$ of the language $\mathcal{L}$.

8 Acknowledgments.

I am indebted to Stuart Allen, Micah Beck, Robert Constable, Jim Hook, Doug Howe, and Garrel Pottinger for many helpful discussions and for and constructive comments on drafts of this paper.

9 Appendix

A Technical results.

This appendix contains proofs of technical results to support claims made in Sections 4 and 5. Perhaps the most important is Theorem 14, which shows Exp^{Δ} to be an acceptable expansion. With the notorious pitfalls involved in reasoning about variables and substitution in mind, an attempt has been made to be very thorough in working out the details of these proofs. We beg the reader's indulgence.

Definition 13 defined *abs* as $\mathrm{abs}_v^\sigma[I] \equiv \lambda \mathrm{scope}_v^\sigma[I].\mathrm{sub}_v^\sigma[I]$. In the proofs of this appendix it will be useful to use the following equivalent definition where abs_v^σ is defined by induction on σ:

$$\begin{array}{ll} \mathrm{abs}_v^v[I] & \equiv I \\ \mathrm{abs}_v^{\langle\sigma_1,\sigma_2\rangle}[\langle I_1, I_2\rangle] & \equiv \mathrm{abs}_v^{\sigma_i}[I_i] \quad (v \in \mathrm{Evars}[\sigma_i],\ i = 1 \text{ or } i = 2) \\ \mathrm{abs}_v^{\lambda y.\sigma_1}[\lambda x.I_1] & \equiv \lambda x.\mathrm{abs}_v^{\sigma_1}[I_1] \end{array}$$

We also define

Definition 23 (trans_v^σ) *Given an s-term σ and a variable $v \in \mathrm{Evars}[\sigma]$ and any expression $e \in \tau[\sigma]$ define the expression $\mathrm{trans}_v^\sigma[e]$ by induction on the structure of σ:*

$$\begin{array}{ll} \mathrm{trans}_v^v[e] & \equiv e \\ \mathrm{trans}_v^{\langle\sigma_1,\sigma_2\rangle}[e] & \equiv \mathrm{trans}_v^{\sigma_i}[\pi_i(e)] \quad (v \in \mathrm{Evars}[\sigma_i],\ i = 1 \text{ or } i = 2) \\ \mathrm{trans}_v^{\lambda y.\sigma_1}[e] & \equiv \lambda y.\mathrm{trans}_v^{\sigma_1}[e(y)] \end{array}$$

Definition 24 ($\hat{\mathcal{F}}_e^\sigma$) *Let σ be an s-term with* $\mathrm{Evars}[\sigma] = \{v_1 \ldots v_n\}$ *and let e be any expression. Let* $z = \mathrm{NewVar}[\mathrm{Vars}[\sigma] \cup \mathrm{Vars}[e]]$. *Define the expression*

$$\hat{\mathcal{F}}_e^\sigma \equiv \lambda z^{\tau[\sigma]}.e \left\{ \begin{array}{ccc} f_{v_1}^\sigma(z)(\vec{y_1}) & \ldots & f_{v_n}^\sigma(z)(\vec{y_n}) \\ \hline v_1 & & v_n \end{array} \right\}$$

where $\vec{y_i} \equiv scope_{v_i}^\sigma[\sigma]$.

We will need the following facts which are easy to prove.

Fact 1

1. $f_v^\sigma \equiv \lambda z^{\tau[\sigma]}.\lambda \mathrm{scope}_v^\sigma[\sigma].\Phi_v^\sigma[z] \equiv \lambda z^{\tau[\sigma]}.\mathrm{trans}_v^\sigma[z]$,
2. $\hat{\mathcal{F}}_e^\sigma =_\Lambda \mathcal{F}_e^\sigma$.

Fact 2 *Let σ be an s-term and I_1, I_2 be σ-instances. If $I_1 =_\alpha I_2$ then then* $\mathrm{abs}_v^\sigma[I_1] =_\alpha \mathrm{abs}_v^\sigma[I_2]$.

Fact 3 *Let σ, σ' be an s-terms with $\sigma =_\alpha \sigma'$ and let I be a σ-instances, then* $\mathrm{abs}_v^\sigma[I] \equiv \mathrm{abs}_v^{\sigma'}[I]$.

Fact 4 *Let e_1 and e_2 be expressions with $e_1 \twoheadrightarrow e_2$, and let σ be an s-term with variable $v \in \mathrm{Evars}[\sigma]$, then* $\mathrm{trans}_v^\sigma[e_1] \twoheadrightarrow \mathrm{trans}_v^\sigma[e_2]$.

Fact 5 *For any expression e and any $n \geq 1$ and expressions $e_1, \ldots, e_n$ and $e_1', \ldots, e_n'$ with $e_i \twoheadrightarrow e_i$ for $i \in \{1, \ldots, n\}$ we have*

$$e\left[\begin{array}{ccc} e_1 & \ldots & e_n \\ \hline v_1 & & v_n \end{array}\right] \twoheadrightarrow e\left[\begin{array}{ccc} e_1' & \ldots & e_n' \\ \hline v_1 & & v_n \end{array}\right]$$

Fact 6 *For any expression e and e' with $e =_\alpha e'$ and any $n \geq 1$ and expressions $e_1, \ldots, e_n$ we have*

$$e\left[\begin{array}{ccc} e_1 & \ldots & e_n \\ \hline v_1 & & v_n \end{array}\right] =_\alpha e'\left[\begin{array}{ccc} e_1 & \ldots & e_n \\ \hline v_1 & & v_n \end{array}\right]$$

The following lemma tells us that under certain circumstances substitution commutes with $\mathrm{trans}_v^\sigma[e]$.

Lemma 6 *For all expressions e_1, e_2 and s-terms σ and variables z with*

1. $\mathrm{Fvars}[e_1] \cap \mathrm{Bvars}[\sigma] = \phi$
2. $z \notin \mathrm{Bvars}[e_1]$ *and* $z \in \mathrm{Fvars}[e_1]$

we have $\mathrm{trans}_v^\sigma[e_1][e_2/z] \equiv \mathrm{trans}_v^{\sigma'}[e_1[e_2/z]]$ *for some s-term $\sigma' =_\alpha \sigma$ with* $\mathrm{Fvars}[e_2] \cap \mathrm{Bvars}[\sigma'] = \phi$.

Proof. By induction on the structure of σ.
Case 1: $\sigma \equiv v^\tau$ then $\mathrm{trans}_v^v[e_1][e_2/z] \equiv e_1[e_2/z] \equiv \mathrm{trans}_v^v[e_1[e_2/z]]$ by the definitions of trans_v^σ.
Case 2: $\sigma \equiv \langle \sigma_1, \sigma_2 \rangle$ then $v \in Vars[\sigma_1]$ or $v \in Vars[\sigma_2]$. We will only consider the former case since the latter can be treated symmetrically. Since I is a σ-instance, $I \equiv \langle I_1, I_2 \rangle$ for some σ_1-instance I_1 and some σ_2-instance I_2. Then

$$\begin{aligned}
\mathrm{trans}_v^{\langle \sigma_1, \sigma_2 \rangle}[e_1][e_2/z] &\equiv \mathrm{trans}_v^{\sigma_1}[\pi_1(e_1)][e_2/z] \\
&\equiv \mathrm{trans}_v^{\sigma_1'}[\pi_1(e_1)[e_2/z]] \qquad \text{(by induction)} \\
&\equiv \mathrm{trans}_v^{\sigma_1'}[\pi_1(e_1[e_2/z])] \\
&\equiv \mathrm{trans}_v^{\langle \sigma_1', \sigma_2' \rangle}[e_1[e_2/z]] \qquad \text{(for any } \sigma_2' =_\alpha \sigma_2)
\end{aligned}$$

Case 3: $\sigma \equiv \lambda y.\sigma_1$ and $I \equiv \lambda x.I_1$ for some σ_1-instance I_1. Then by the definition of trans_v^σ

$$\mathrm{trans}_v^{\lambda y.\sigma_1}[e_1][e_2/z] \equiv (\lambda y.\mathrm{trans}_v^{\sigma_1}[e_1(y)])[e_2/z],$$

which leaves three cases to consider:
Case 3.1: $y = z$. This cannot occur by condition 2.
Case 3.2: $y \notin \mathrm{Fvars}[e_2]$. Then

$$\begin{aligned}
&\equiv \lambda y.\mathrm{trans}_v^{\sigma_1}[e_1(y)][e_2/z] \\
&\equiv \lambda y.\mathrm{trans}_v^{\sigma_1'}[(e_1(y))[e_2/z]] \quad \text{(by induction)} \\
&\equiv \lambda y.\mathrm{trans}_v^{\sigma_1'}[e_1[e_2/z](y)] \quad \text{(since } y \neq z \text{ by condition 2)} \\
&\equiv \mathrm{trans}_v^{\lambda y.\sigma_1'}[e_1[e_2/z]]
\end{aligned}$$

Case 3.3: $y \in \mathrm{Fvars}[e_2]$. Then let w be a new variable so that

$$\begin{aligned}
&\equiv \lambda w.\mathrm{trans}_v^{\sigma_1}[e_1(y)][w/y][e_2/z] \\
&\equiv \lambda w.\mathrm{trans}_v^{\sigma_1}[e_1(w)][e_2/z] \\
&\equiv \lambda w.\mathrm{trans}_v^{\sigma_1'}[(e_1(w))[e_2/z]] \quad \text{(by induction)} \\
&\equiv \lambda w.\mathrm{trans}_v^{\sigma_1'}[e_1[e_2/z](w)] \\
&\quad \text{(since } w \neq z \text{ by condition choice of } w) \\
&\equiv \mathrm{trans}_v^{\lambda w.\sigma_1'}[e_1[e_2/z]]
\end{aligned}$$

□

The following lemma tells us that $\mathrm{trans}_v^\sigma[I]$ does indeed extract the expression $\mathrm{abs}_v^\sigma[I]$ from a σ-instance I by reduction.

Lemma 7 *If σ is an s-term with $v \in \mathrm{Evars}[\sigma]$ and I is a σ-instance with $\mathrm{Fvars}[I] \cap \mathrm{Bvars}[\sigma] = \phi$ then $\mathrm{trans}_v^\sigma[I] \twoheadrightarrow \mathrm{abs}_v^\sigma[I]$.*

Proof. By induction on the structure of σ.
Case 1: $\sigma \equiv v^\tau$. Then $\mathrm{trans}_v^v[I] \equiv I \equiv \mathrm{abs}_v^v[I]$ by the definitions of trans_v^σ and abs_v^σ.

Case 2: $\sigma \equiv \langle \sigma_1, \sigma_2 \rangle$. Then $v \in Vars[\sigma_1]$ or $v \in Vars[\sigma_2]$. We will only consider the former case since the latter can be treated symmetrically. Since I is a σ-instance, $I \equiv \langle I_1, I_2 \rangle$ for some σ_1-instance I_1 and some σ_2-instance I_2 we have

$$\begin{array}{lll}
\mathrm{trans}_v^{\sigma}[I] \equiv & \mathrm{trans}_v^{\langle \sigma_1, \sigma_2 \rangle}[\langle I_1, I_2 \rangle] & \\
\equiv & \mathrm{trans}_v^{\sigma_1}[\pi_1(\langle I_1, I_2 \rangle)] & \\
\longrightarrow & \mathrm{trans}_v^{\sigma_1}[I_1] & \text{(by Fact 4)} \\
\longrightarrow\!\!\!\!\!\rightarrow & \mathrm{abs}_v^{\sigma_1}[I_1] & \text{(by induction)} \\
\equiv & \mathrm{abs}_v^{\langle \sigma_1, \sigma_2 \rangle}[\langle I_1, I_2 \rangle] & \\
\equiv & \mathrm{abs}_v^{\sigma}[I]. &
\end{array}$$

Case 3: $\sigma \equiv \lambda y.\sigma_1$. Then $I \equiv \lambda x.I_1$ for some σ_1-instance I_1 and

$$\begin{array}{lll}
\mathrm{trans}_v^{\sigma}[I] \equiv & \mathrm{trans}_v^{\lambda y.\sigma_1}[\lambda x.I_1] & \\
\equiv & \lambda y.\mathrm{trans}_v^{\sigma_1}[(\lambda x.I_1)(y)] & \\
\longrightarrow & \lambda y.\mathrm{trans}_v^{\sigma_1}[I_1[y/x]] & \text{(by Fact 4)} \\
\longrightarrow\!\!\!\!\!\rightarrow & \lambda y.\mathrm{abs}_v^{\sigma_1}[I_1[y/x]] & \text{(by induction)} \\
\equiv & \mathrm{abs}_v^{\lambda y.\sigma_1}[\lambda y.I_1[y/x]] & \\
=_{\alpha} & \mathrm{abs}_v^{\lambda y.\sigma_1}[\lambda x.I_1[y/x][x/y]] & \text{(by Fact 3)} \\
\equiv & \mathrm{abs}_v^{\lambda y.\sigma_1}[\lambda x.I_1] & (\mathrm{Fvars}[I] \cap \mathrm{Bvars}[\sigma] = \phi) \\
\equiv & \mathit{absv}\sigma[I]. &
\end{array}$$

□

Lemma 8 *(Lemma 1) If σ is an s-term with $v \in \mathrm{Evars}[\sigma]$ and I is a σ-instance then*

$$f_v^{\sigma}(I) \quad \longrightarrow\!\!\!\!\!\rightarrow \quad \mathrm{abs}_v^{\sigma}[I].$$

Proof.

$$\begin{array}{lll}
f_v^{\sigma}(I) \equiv & (\lambda z.\mathrm{trans}_v^{\sigma}[z])(I) & \\
\longrightarrow & \mathrm{trans}_v^{\sigma}[z][I/z] & \\
\equiv & \mathrm{trans}_v^{\sigma'}[I] & \text{(by Lemma 6)} \\
\longrightarrow\!\!\!\!\!\rightarrow & \mathrm{abs}_v^{\sigma'}[I] & \text{(by Lemma 7)} \\
\equiv & \mathrm{abs}_v^{\sigma}[I] & \text{(by Fact 3)}
\end{array}$$

□

Before proving that Exp^{Δ} is an acceptable expansion we will need a few more technical lemmas.

Lemma 9 *Given an expression e, an $n \geq 1$, a set $\{v_1, \ldots, v_n\}$ of pairwise distinct variables, a collection of variable vectors $\vec{y_1}, \ldots, \vec{y_n}$ (each $\vec{y_i}$ possibly null), and any variable z with*

1. $z \notin \{\vec{v}\}$

2. $x \notin \{\vec{v}\} \cup (\bigcup_{1 \leq i \leq n} \{\vec{y_i}\})$
3. $\text{Bvars}[e] \cap \{\vec{v}\} = \phi$
4. $z \notin \text{Bvars}[e]$

we have

$$e\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \equiv e \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}.$$

Proof. By induction on the structure of e.
Case 1: The result is trivial if e is a constant.
Case 2: If e is a variable, $e \equiv v$, the there are two cases to consider.
Case2.1: $v \notin \{\vec{v}\}$. Then

$$\begin{aligned} & v\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \\ & \quad \equiv \quad v \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \\ & \quad \equiv \quad v \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \\ & \qquad \text{(since } z \notin \textstyle\bigcup_{1 \leq i \leq n} \text{Fvars}[v_i(\vec{y_i})] \text{ by conditions 1 and 2.)} \end{aligned}$$

Case2.2: $v = v_j$ for some $j \in \{1, \ldots, n\}$. Then

$$\begin{aligned} & v_j\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \\ & \quad \equiv \quad (v_j(\vec{y_j})) \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \\ & \quad \equiv \quad (v_j(\vec{y_j})) \quad (\text{since } x \notin \{\vec{v}\} \cup \{\vec{y}\}) \\ & \quad \equiv \quad v_j \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \quad (\text{since } x \notin \{\vec{v}\}) \\ & \quad \equiv \quad v_j \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \end{aligned}$$

Cases 3, 4, and 5: e is of the form $\pi_i(e')$, $\langle e'', e'' \rangle$, or $e'(e'')$. The results go through easily since substitution is defined by a simple induction in these cases and the conditions $1 - -4$ still hold.
Case 6: $e \equiv \lambda w.e'$. Following the last clause in the definition of substitution we can see that there is only the case of $x \notin \vec{v}$ to consider (since $w \in \vec{v}$. by condition 3), so that

$$(\lambda w.e')\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}\left[\begin{array}{c} \frac{z}{x} \end{array} \right]$$

$$\equiv \quad (\lambda w.e'\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\})\left[\begin{array}{c} \frac{z}{x} \end{array} \right]$$

at which point there are three cases to consider (again since $w \notin \{\vec{v}\}$):
Case 6.1 : $w = x$. Then

$$\equiv \quad (\lambda w.e'\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\})\left[\begin{array}{c} \frac{z}{x} \end{array} \right]$$

$$\equiv \quad (\lambda w.e'\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\})$$

$$\equiv \quad (\lambda w.e')\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}$$

$$\equiv \quad (\lambda w.e')\left[\begin{array}{c} \frac{z}{x} \end{array} \right]\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}$$

Case 6.2 : $w \neq x$ and $w \neq z$. Then

$$\equiv \quad (\lambda w.e'\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\})\left[\begin{array}{c} \frac{z}{x} \end{array} \right]$$

$$\equiv \quad (\lambda w.e'\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}\left[\begin{array}{c} \frac{z}{x} \end{array} \right])$$

$$\equiv \quad (\lambda w.e'\left[\begin{array}{c} \frac{z}{x} \end{array} \right]\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\})$$

(by induction since conditions 1-4 still hold)

$$\equiv \quad (\lambda w.e')\left[\begin{array}{c} \frac{z}{x} \end{array} \right]\left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}$$

Case 6.3: $w \neq x$ and $w = z$. This cannot occur by condition 4. □

Recall that $\mathcal{V} = \mathcal{V}_s \cup \mathcal{V}_r$ where $\mathcal{V}_s \cap \mathcal{V}_r = \phi$.

Lemma 10 *Given an expression e, an $n \geq 1$, a set $\{\vec{v}\} = \{v_1, \ldots, v_n\}$ of pairwise distinct variables, and any collection of variable vectors $\vec{y_1}, \ldots, \vec{y_n}$ (each $\vec{y_i}$ possibly null) with*

1. $\{\vec{y_i}\} \subseteq \{\vec{y}\}$ *for* $i \in \{1, \ldots, n\}$
2. $\{\vec{y}\} \cap (\bigcup_{1 \leq i \leq n} \mathrm{Fvars}[\lambda \vec{y_i}.e_i]) = \phi$
3. $\mathrm{Bvars}[e] \cap \{\vec{v}\} = \phi$
4. $\{\vec{v}\} \subset \mathcal{V}_s$

we have

$$e\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{\lambda\vec{y_1}.e_1}{v_1} & \cdots & \frac{\lambda\vec{y_n}.e_n}{v_n}\end{array}\right]$$

$$\longrightarrow\!\!\!\!\!\rightarrow\ e\left[\begin{array}{ccc}\frac{(e_1,\{\vec{y_1}\})}{v_1} & \cdots & \frac{(e_n,\{\vec{y_n}\})}{v_n}\end{array}\right]$$

Proof. By induction on the structure of e.
Case 1: The result is trivial if e is a constant.
Case 2: e is a variable, $e \equiv v$, which involves two subcases to consider. Case2.1: $v \notin \{\vec{v}\}$. Trivial. Case 2.2: $v = v_j$ for some $j \in \{1,\ldots,n\}$. Then

$$\begin{aligned} & v_j\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{\lambda\vec{y_1}.e_1}{v_1} & \cdots & \frac{\lambda\vec{y_n}.e_n}{v_n}\end{array}\right] \\ & \equiv \quad (\lambda\vec{y_j}.e_j)(\vec{y_j}) \\ & \longrightarrow\!\!\!\!\!\rightarrow \quad e_i \\ & \equiv \quad v_j\left[\begin{array}{ccc}\frac{(e_1,\{\vec{y_1}\})}{v_1} & \cdots & \frac{(e_n,\{\vec{y_n}\})}{v_n}\end{array}\right] \end{aligned}$$

Cases 3, 4, and 5: e is of the form $\pi_i(e')$, $\langle e'', e''\rangle$, or $e'(e'')$. The results go through easily since substitution is defined by a simple induction in these cases and the conditions $1-4$ still hold.
Case 6: $e \equiv \lambda x.e'$. Following the last clause in the definition of substitution we can see that there is only the case of $x \notin \vec{v}$ to consider (since $x \in \vec{v}$. by condition 3), so that

$$\begin{aligned} & (\lambda x.e')\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{\lambda\vec{y_1}.e_1}{v_1} & \cdots & \frac{\lambda\vec{y_n}.e_n}{v_n}\end{array}\right] \\ & \equiv \quad (\lambda x.e'\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\})\left[\begin{array}{ccc}\frac{\lambda\vec{y_1}.e_1}{v_1} & \cdots & \frac{\lambda\vec{y_n}.e_n}{v_n}\end{array}\right] \end{aligned}$$

at which point there are two cases to consider (again since $x \notin \{\vec{v}\}$):
Case 6.1 : $x \notin \bigcup_{1\le i\le n} \mathrm{Fvars}[\lambda\vec{y_i}.e']$.

$$\begin{aligned} & \equiv \quad (\lambda x.e'\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{\lambda\vec{y_1}.e_1}{v_1} & \cdots & \frac{\lambda\vec{y_n}.e_n}{v_n}\end{array}\right]) \\ & \longrightarrow\!\!\!\!\!\rightarrow \quad (\lambda x.e'\left[\begin{array}{ccc}\frac{(e_1,\{\vec{y_1}\})}{v_1} & \cdots & \frac{(e_n,\{\vec{y_n}\})}{v_n}\end{array}\right]) \quad \text{(by induction)} \\ & \equiv \quad (\lambda x.e')\left[\begin{array}{ccc}\frac{(e_1,\{\vec{y_1}\})}{v_1} & \cdots & \frac{(e_n,\{\vec{y_n}\})}{v_n}\end{array}\right] \end{aligned}$$

since $x \notin \bigcup_{1\le i\le n} \mathrm{Fvars}[\lambda\vec{y_i}.e_1] = \bigcup_{1\le i\le n}(\mathrm{Fvars}[e_1] - \{\vec{y_i}\})$

Case 6.2 :$x \in \bigcup_{1\le i\le n} \mathrm{Fvars}[\lambda \vec{y_i}.e_1]$. Looking at the expression

$$(\lambda x.e' \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \ldots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}) \left[\begin{array}{ccc} \frac{\lambda \vec{y_1}.e_1}{v_1} & \ldots & \frac{\lambda \vec{y_n}.e_n}{v_n} \end{array} \right]$$

we see that there are three subcases to consider.
Case 6.2.1 : $x \in \{\vec{y}\}$: This cannot occur by condition 2.
Case 6.2.2 : $x \notin \{\vec{y}\}$ and $z \in \{\vec{v}\}$. This cannot occur by condition 4.
Case 6.2.3 :$x \notin \{\vec{y}\}$ and $z \notin \{\vec{v}\}$.

$$(\lambda x.e' \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \ldots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\}) \left[\begin{array}{ccc} \frac{\lambda \vec{y_1}.e_1}{v_1} & \ldots & \frac{\lambda \vec{y_n}.e_n}{v_n} \end{array} \right]$$

$$\equiv \quad \lambda z.e' \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \ldots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \left[\begin{array}{ccc} \frac{\lambda \vec{y_1}.e_1}{v_1} & \ldots & \frac{\lambda \vec{y_n}.e_n}{v_n} \end{array} \right]$$

$$\equiv \quad \lambda z.e' \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \left\{ \begin{array}{ccc} \frac{v_1(\vec{y_1})}{v_1} & \ldots & \frac{v_n(\vec{y_n})}{v_n} \end{array} \right\} \left[\begin{array}{ccc} \frac{\lambda \vec{y_1}.e_1}{v_1} & \ldots & \frac{\lambda \vec{y_n}.e_n}{v_n} \end{array} \right]$$

by Lemma 9 since $z \notin \{\vec{v}\}$ by $\{\vec{v}\} \subset \mathcal{V}_s$

and $z \notin \mathrm{Bvars}[e]$ by the definition of substitution.

$$\longrightarrow \quad (\lambda z.e' \left[\begin{array}{c} \frac{z}{x} \end{array} \right] \left[\begin{array}{ccc} \frac{(e_1, \{\vec{y_1}\})}{v_1} & \ldots & \frac{(e_n, \{\vec{y_n}\})}{v_n} \end{array} \right])$$

by induction (condition 3 still holds since $\{\vec{v}\}$ are special)

$$\equiv \quad (\lambda z.e' \left[\begin{array}{c} \frac{z}{x} \end{array} \right]) \left[\begin{array}{ccc} \frac{(e_1, \{\vec{y_1}\})}{v_1} & \ldots & \frac{(e_n, \{\vec{y_n}\})}{v_n} \end{array} \right]$$

since $z \notin \bigcup_{1\le i\le n} \mathrm{Fvars}[\lambda \vec{y_i}.e_1] = \bigcup_{1\le i\le n} (\mathrm{Fvars}[e_1] - \{\vec{y_i}\})$

$$=_\alpha \quad (\lambda x.e') \left[\begin{array}{ccc} \frac{(e_1, \{\vec{y_1}\})}{v_1} & \ldots & \frac{(e_n, \{\vec{y_n}\})}{v_n} \end{array} \right]$$

□

The following lemma shows that simultaneous α-conversion is well defined.

Lemma 11 *Let e be an expression. Given any $n \ge 1$ and variables*

$$\begin{array}{rcl} \vec{v} & \equiv & [v_1, v_2, \ldots v_n] \\ \vec{w} & \equiv & [w_1, w_2, \ldots w_n] \end{array}$$

if $\{\vec{v}\} \cap \{\vec{w}\} = \phi$ and $\mathrm{Fvars}[e] \cap \{\vec{w}\} = \phi$ then $e[\vec{w}/\vec{v}]^\alpha =_\alpha e$.

Proof. By induction on the structure of e. All cases are trivial except for $e \equiv \lambda x.e_1$ which involves looking at three subcases:
Case 1: $x \notin \{\vec{v}\} \cup \{\vec{w}\}$. Since by assumption $\mathrm{Fvars}[\lambda x.e_1] \cap \{\vec{w}\} = \phi$ and $x \notin \{\vec{w}\}$

we have $\text{Fvars}[e_1] \cap \{\vec{w}\} = \phi$ which allows us to use the induction hypothesis on the expression $e_1[\vec{w}/\vec{v}]^\alpha$ in

$$\begin{array}{lll} (\lambda x.e_1)[\vec{w}/\vec{v}]^\alpha & \equiv & \lambda x.e_1[\vec{w}/\vec{v}]^\alpha \\ & =_\alpha & \lambda x.e_1 \quad (e_1[\vec{w}/\vec{v}]^\alpha =_\alpha e_1 \text{ by induction}) \\ & \equiv & e. \end{array}$$

Case 2: $x \notin \{\vec{w}\}$ and $x = v_j$ for some $j \in \{1,\ldots,n\}$. We are again allowed to use the induction hypothesis on the expression $e_1[\vec{w}/\vec{v}]^\alpha$ (for the same reasons as cited in Case 1) in

$$\begin{array}{lll} (\lambda x.e_1)[\vec{w}/\vec{v}]^\alpha & \equiv & (\lambda w_j.e_1[\vec{w}/\vec{v}]^\alpha[w_j/x]) \\ & =_\alpha & \lambda w_j.e_1[w_j/x] \quad (e_1[\vec{w}/\vec{v}]^\alpha =_\alpha e_1 \text{ by induction}) \\ & =_\alpha & \lambda x.e_1 \\ & \equiv & e. \end{array}$$

Case 3: $x \notin \{\vec{v}\}$ and $x = w_j$ for some $j \in \{1,\ldots,n\}$. By assumption $\text{Fvars}[\lambda x.e_1] \cap \{\vec{w}\} = \phi$ so if z is a new variable with $z \notin \text{Fvars}[e_1] \cup \{\vec{w}\}$ we have $\text{Fvars}[e_1[z/x]] \cap \{\vec{w}\} = \phi$ which allows us to use the induction hypothesis on the expression $e_1[z/x]$ in

$$\begin{array}{lll} (\lambda x.e_1)[\vec{w}/\vec{v}]^\alpha & \equiv & (\lambda z.e_1[z/x][\vec{w}/\vec{v}]^\alpha) \\ & =_\alpha & \lambda z.e_1[z/x] \\ & & (e_1[z/x][\vec{w}/\vec{v}]^\alpha =_\alpha e_1[z/x] \text{ by induction}) \\ & =_\alpha & \lambda x.e_1 \\ & \equiv & e. \end{array}$$

□

Fact 7 *Let $\vec{y} = [y_1,\ldots,y_n]$ be pairwise distinct variables, and e be any expression. If $y_i \notin \text{Fvars}[e]$ then $e[\vec{e}/\vec{y}] =_\alpha e[\vec{e}/\vec{y}]/\{y_i\}$.*

Fact 8 *Let $\vec{y} = [y_1,\ldots,y_n]$ and $\vec{v} = [v_1,\ldots,v_n]$ be pairwise distinct variables, $\vec{e_1} = [e_{1,1},\ldots,e_{1,n}]$ and $\vec{e_2} = [e_{2,1},\ldots,e_{2,n}]$ be expressions, then*

$$e[\vec{e_1}/\vec{y}][\vec{e_2}/\vec{v}] =_\alpha e[\vec{e_3}/\vec{w}]$$

where

$$\begin{array}{lll} \vec{e_3} & = & [e_{1,1}[\vec{e_2}/\vec{v}],\ldots,e_{1,n}[\vec{e_2}/\vec{v}],e_{2,1},\ldots,e_{2,n}] \\ \vec{w} & = & [y_1,\ldots,y_n,v_1,\ldots,v_n]. \end{array}$$

The following lemma is of no general interest but allows us to obtain Corollary 13 which in turn is justifies one step in the proof of Theorem 14.

Lemma 12 *For any expression e and any $k \geq 1$ pairwise distinct variables*

$$\vec{y} \equiv [y_1, y_2, \ldots y_k]$$

and any function s mapping variables to variables let

$$\vec{u} \equiv [u_1, u_2, \ldots u_k]$$

where $u_i = s(y_i)$. Then if

1. $\{\vec{y}\} \cap \{\vec{u}\} = \{\vec{y}\} \cap \{\vec{v}\} = \{\vec{v}\} \cap \{\vec{u}\} = \phi$
2. Bvars$[e_i] = \phi$
3. Vars$[e_i]$ (= Evars$[e_i]$) $= \{\vec{y_i}\} \cup \{v_i\}$ *where* $\{\vec{y_i}\} \subseteq \{\vec{y}\}$.

we have

$$e\left\{ \begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array} \right\} \left[\begin{array}{c} \vec{u} \\ \vec{y} \end{array} \right]^{\alpha} \left[\begin{array}{c} \vec{u} \\ \vec{y} \end{array} \right] =_{\alpha}$$

$$e\left[\begin{array}{c} \vec{u} \\ \vec{y} \end{array} \right]^{\alpha} \left[\begin{array}{c} \vec{u} \\ \vec{y} \end{array} \right] \left\{ \begin{array}{ccc} e_1[\vec{u}/\vec{y}] & \cdots & e_n[\vec{u}/\vec{y}] \\ \overline{v_1} & & \overline{v_n} \end{array} \right\}.$$

Proof. By induction on the structure of e: Case 1: The result is trivial if e is a constant.
Case 2: e is a variable, $e \equiv x$, which involves three subcases to consider.
Case2.1: $x \in \vec{v}$, say $x \equiv v_j$. Then both sides of the equation evaluate to $e_j[\vec{u}/\vec{y}]$ since $\vec{y} \cap \vec{v} = \phi$ and since Bvars$[e_j] = \phi$ it is easy to see from the definition of simultaneous α-conversion that $e_j[\vec{u}/\vec{y}]^{\alpha} \equiv e_j$.
Case 2.2: $x \in \vec{y}$, say $x \equiv y_j$. Then both sides of the equation evaluate to u_j since $\vec{y} \cap \vec{u} \cap \vec{v} = \phi$.
Case 2.3: $x \notin \vec{y} \cup \vec{u}$. Then both sides of the equation evaluate to x.
Cases 3, 4, and 5: e is of the form $\pi_i(e')$, $\langle e'', e'' \rangle$, or $e'(e'')$. The result goes through easily since substitution and simultaneous α-conversion are defined by induction and since the conditions still hold.
Case 6: $e \equiv \lambda x.e_0$. There are two subcases to consider.
Case 6.1: $x \equiv y_j$. Evaluating the left side of the equation we get

$$\begin{array}{ll} (\lambda y_j.e_0)\left\{ \begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^{\alpha}[\vec{u}/\vec{y}] & \\ \equiv & (\lambda y_j.e_0\left\{ \begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array} \right\})[\vec{u}/\vec{y}]^{\alpha}[\vec{u}/\vec{y}] \\ \equiv & (\lambda u_j.e_0\left\{ \begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^{\alpha}[u_j/y_j])[\vec{u}/\vec{y}] \\ =_{\alpha} & \lambda u_j.e_0\left\{ \begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^{\alpha}[u_j/y_j][\vec{u}/\vec{y}]/\{y_j\} \\ & \text{(by Fact 7)} \\ =_{\alpha} & \lambda u_j.e_0\left\{ \begin{array}{ccc} e_1 & \cdots & e_n \\ \overline{v_1} & & \overline{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^{\alpha}[\vec{u}/\vec{y}] \\ & \text{(by Fact 8).} \end{array}$$

Now, turning to the right side of the equation we have

$$\begin{array}{ll} (\lambda y_j.e_0)[\vec{u}/\vec{y}]^{\alpha}[\vec{u}/\vec{y}]\left\{ \begin{array}{ccc} e_1[\vec{u}/\vec{y}] & \cdots & e_n[\vec{u}/\vec{y}] \\ \overline{v_1} & & \overline{v_n} \end{array} \right\} & \\ \equiv & (\lambda u_j.e_0[\vec{u}/\vec{y}]^{\alpha}[u_j/y_j])[\vec{u}/\vec{y}]\left\{ \begin{array}{ccc} e_1[\vec{u}/\vec{y}] & \cdots & e_n[\vec{u}/\vec{y}] \\ \overline{v_1} & & \overline{v_n} \end{array} \right\} \\ =_{\alpha} & (\lambda u_j.e_0[\vec{u}/\vec{y}]^{\alpha}[u_j/y_j][\vec{u}/\vec{y}]/\{y_j\})\left\{ \begin{array}{ccc} e_1[\vec{u}/\vec{y}] & \cdots & e_n[\vec{u}/\vec{y}] \\ \overline{v_1} & & \overline{v_n} \end{array} \right\} \\ & \text{(by Fact 7)} \end{array}$$

$$=_\alpha \quad (\lambda u_j.e_0[\vec{u}/\vec{y}]^\alpha[\vec{u}/\vec{y}])\left\{ \begin{array}{ccc} \frac{e_1[\vec{u}/\vec{y}]}{v_1} & \cdots & \frac{e_n[\vec{u}/\vec{y}]}{v_n} \end{array} \right\}$$

(by Fact 8)

$$\equiv \quad \lambda u_j.e_0[\vec{u}/\vec{y}]^\alpha[\vec{u}/\vec{y}]\left\{ \begin{array}{ccc} \frac{e_1[\vec{u}/\vec{y}]}{v_1} & \cdots & \frac{e_n[\vec{u}/\vec{y}]}{v_n} \end{array} \right\}.$$

which is $=_\alpha$ to the result on the left by induction.
Case 6.2: $x \equiv u_j$. Evaluating the left side of the equation we get

$$\begin{aligned} &(\lambda u_j.e_0)\left\{ \begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^\alpha[\vec{u}/\vec{y}] \\ &\quad \equiv \quad (\lambda u_j.e_0\left\{ \begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array} \right\})[\vec{u}/\vec{y}]^\alpha[\vec{u}/\vec{y}] \\ &\quad \equiv \quad (\lambda z_1.e_0\left\{ \begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array} \right\}[z_1/u_j][\vec{u}/\vec{y}]^\alpha)[\vec{u}/\vec{y}] \\ &\quad \equiv \quad (\lambda z_1.e_0[z_1/u_j]\left\{ \begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^\alpha)[\vec{u}/\vec{y}] \\ &\quad \text{since } \vec{u} \cap \text{Vars}[e_i] = \phi \\ &\quad \equiv \quad \lambda z_1.(e_0[z_1/u_j])\left\{ \begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array} \right\}[\vec{u}/\vec{y}]^\alpha[\vec{u}/\vec{y}] \end{aligned}$$

where $z_1 = \text{NewVar}[\{\vec{u}\} \cup \{\vec{y}\} \cup \text{Vars}[e_0\left\{ \begin{array}{ccc} \frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n} \end{array} \right\}]]$. Now, turning to the right side of the equation we have

$$\begin{aligned} &(\lambda u_j.e_0)\left[\begin{array}{c} \vec{u} \\ \vec{y} \end{array} \right]^\alpha [\vec{u}/\vec{y}]\left\{ \begin{array}{ccc} \frac{e_1[\vec{u}/\vec{y}]}{v_1} & \cdots & \frac{e_n[\vec{u}/\vec{y}]}{v_n} \end{array} \right\} \\ &\quad \equiv \quad (\lambda z_2.e_0\left[\begin{array}{c} z_2 \\ u_j \end{array} \right][\vec{u}/\vec{y}]^\alpha)[u_j/y_j])[\vec{u}/\vec{y}]\left\{ \begin{array}{ccc} \frac{e_1[\vec{u}/\vec{y}]}{v_1} & \cdots & \frac{e_n[\vec{u}/\vec{y}]}{v_n} \end{array} \right\} \\ &\quad \equiv \quad \lambda z_2.(e_0[z_2/u_j])[\vec{u}/\vec{y}]^\alpha[u_j/y_j])[\vec{u}/\vec{y}]\left\{ \begin{array}{ccc} \frac{e_1[\vec{u}/\vec{y}]}{v_1} & \cdots & \frac{e_n[\vec{u}/\vec{y}]}{v_n} \end{array} \right\}. \end{aligned}$$

where $z_2 = \text{NewVar}[\{\vec{u}\} \cup \{\vec{y}\} \cup \text{Vars}[e_0]]$. But $z_1 = z_2$ since $\text{Vars}[e_i] \subseteq (\vec{v} \cup \vec{y})$ by condition 3 so that the left and right-hand sides of the equation are $=_\alpha$ by induction. □.

Corollary 13 *For any expression e and any $k \geq 1$ pairwise distinct variables*

$$\vec{y} \quad \equiv \quad [y_1, y_2, \ldots y_k]$$

and any function s mapping variables to variables let

$$\vec{u} \quad \equiv \quad [u_1, u_2, \ldots u_k]$$

where $u_i = s(y_i)$. Then if

1. $\{\vec{y}\} \cap \{\vec{u}\} = \phi$
2. $\{\vec{y_i}\} \subseteq \{\vec{y}\}$
3. *every occurrence of v_i in e is in the scope of each variable in $\vec{y_i}$*

we have

$$e\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{c}\vec{u}\\ \vec{y}\end{array}\right]^{\alpha} =_{\alpha} e\left[\begin{array}{c}\vec{u}\\ \vec{y}\end{array}\right]^{\alpha}\left\{\begin{array}{ccc}\frac{v_1(\vec{w_1})}{v_1} & \cdots & \frac{v_n(\vec{w_n})}{v_n}\end{array}\right\}$$

where $\vec{w_i} = s(\vec{y_i})$.

We are now ready to prove the main theorem:

Theorem 14 *(Theorem 5)* Exp^{Δ} *is an acceptable expansion.*

Proof. Assume that

$$d(\sigma) \triangleq e$$

is a well-formed Δ-equation and that I is a σ-instance. Let $I' = \mathrm{ConvInst}[I]$ and let

$$\begin{array}{lcl}\vec{y_i} & \equiv & \mathrm{scope}^{\sigma}_{v_i}[\sigma]\\ \mathrm{abs}^{\sigma}_{v_i}[I'] & \equiv & \lambda\mathrm{scope}^{\sigma}_{v}[I'].\mathrm{sub}^{\sigma}_{v_i}[I'] \equiv \lambda\vec{w_i}.\mathrm{sub}^{\sigma}_{v_i}[I']\end{array}$$

then

$$\begin{array}{lll}d(I) =_{\alpha} & d(I') & \\ \longrightarrow & \mathcal{F}^{\sigma}_{e}(I') & \\ =_{\Lambda} & \hat{\mathcal{F}}^{\sigma}_{e}(I') & \\ & \text{(by Fact 1)} & \\ \equiv & (\lambda z.e\left\{\begin{array}{ccc}\frac{f^{\sigma}_{v_1}(z)(\vec{y_1})}{v_1} & \cdots & \frac{f^{\sigma}_{v_n}(z)(\vec{y_n})}{v_n}\end{array}\right\})(I') & \\ \equiv & (\lambda z.e\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{f^{\sigma}_{v_1}(z)}{v_1} & \cdots & \frac{f^{\sigma}_{v_n}(z)}{v_n}\end{array}\right])(I') & \\ \longrightarrow & e\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{f^{\sigma}_{v_1}(I')}{v_1} & \cdots & \frac{f^{\sigma}_{v_n}(I')}{v_n}\end{array}\right] & \\ \longrightarrow\!\!\!\!\rightarrow & e\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{\mathrm{abs}^{\sigma}_{v_1}(I')}{v_1} & \cdots & \frac{\mathrm{abs}^{\sigma}_{v_n}(I')}{v_n}\end{array}\right] & \\ & \text{(By Lemmas 8 and 5)} & \\ =_{\alpha} & e\left\{\begin{array}{ccc}\frac{v_1(\vec{y_1})}{v_1} & \cdots & \frac{v_n(\vec{y_n})}{v_n}\end{array}\right\}\left[\begin{array}{c}\vec{u}\\ \vec{y}\end{array}\right]^{\alpha}\left[\begin{array}{ccc}\frac{\mathrm{abs}^{\sigma}_{v_1}(I')}{v_1} & \cdots & \frac{\mathrm{abs}^{\sigma}_{v_n}(I')}{v_n}\end{array}\right] & \\ & \text{(by Fact 6 and Lemma11 and } \mathrm{Vars}[\sigma] \subset \mathcal{V}_s) & \end{array}$$

$$\equiv \quad e\left[\begin{array}{c}\vec{u}\\ \vec{y}\end{array}\right]^{\alpha}\left\{\begin{array}{ccc}\frac{v_1(\vec{w_1})}{v_1} & \cdots & \frac{v_n(\vec{w_n})}{v_n}\end{array}\right\}\left[\begin{array}{ccc}\frac{e_1}{v_1} & \cdots & \frac{e_n}{v_n}\end{array}\right]$$

(for $e_i \equiv \lambda\vec{w_i}.\mathrm{sub}^{\sigma}_{v_i}[I']$ by Lemma 13)

$$\rightarrowtail\!\!\!\!\rightarrow e\left[\begin{array}{c}\vec{u}\\ \vec{y}\end{array}\right]^{\alpha}\left[\begin{array}{ccc}\frac{(\mathrm{sub}^{\sigma}_{v_1}[I'],\{\vec{w_1}\})}{v_1} & \cdots & \frac{(\mathrm{sub}^{\sigma}_{v_n}[I'],\{\vec{w_n}\})}{v_n}\end{array}\right]$$

(by Lemma 10)

$$\equiv \quad \mathrm{Exp}^{\Delta}[d(I)] \quad \text{(by definition of } \mathrm{Exp}^{\Delta}\text{)}$$

□

References

[AHM87] Arnon Avron, Furio Honsell, and Ian Mason. Using typed lambda calculus to implement formal systems on a machine. LFCS Report Series ECS-LFCS-87-31, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987.

[And86] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, revised edition, 1984.

[Bur86] R.M. Burstall. Research in interactive theorem proving at edinburgh university. LFCS Report Series ECS-LFCS-86-12, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1986.

[CF58] Haskall B. Curry and Robert Feys. *Combinatory Logic*, volume I. North-Holland, Amsterdam, 1958.

[CH85] Thierry Coquand and Gérard Huet. Constuctions: A higher order proof system for mechanizing mathematics. In Bruno Buchberger, editor, *EUROCAL '85*, volume 203 of *Lecture Notes in Computer Science*, pages 151–184. Springer-Verlag, 1985.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Con86] Robert L. Constable, et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[dB70] N. G. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Symposium on Automated Demonstration*, volume 125 of *Lecture Notes in Mathematics*, Berlin, December 1970. Springer-Verlag.

[dB80] N. G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, 1980.

[dV87] R.C. de Vrijer. *Surjective Pairing and Strong Normalization: Two Themes in the Lambda Calculus.* University of Amsterdam, 1987. PhD Thesis.

[GMW79] Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science.* Springer-Verlag, 1979.

[Gri87] Timothy G. Griffin. An environment for formal systems. Technical Report 87-846, Department of Computer Science, Cornell University, 1987. (also LFCS report ECS-LFCS-87-34, Department of Computer Science, University of Edinburgh).

[HHP87] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Proceedings of the Second Symposium on Logic in Computer Science*, 1987.

[HL78] Gérard Huet and Bernard Lang. Proving applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[How86] Douglas J. Howe. Implementing number theory: An experiment with nuprl. Technical Report TR 86–752, Department of Computer Science, Cornell University, 1986.

[How87] Douglas J. Howe. The computational behaviour of girard's paradox. Technical Report TR-87-820, Cornell University, Department of Computer Science, Cornell University, Ithaca, NY 14853, 1987.

[HS86] R. J. Hindley and J.P Seldin. *Introduction to Combinators and λ-Calculus.* London Mathematical Society Student Texts. Cambridge University Press, 1986.

[Hue75] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical computer science*, 1:25–57, 1975.

[Jon87] Claire Jones. The interactive proof editor. Technical Report LFCS-TN-11, Department of Computer Science, University of Edinburgh, 1987.

[KFFD87] Eugene Kohlbecker, Daniel Friedman, Matthias Felleisen, and Bruce Duba. Hygenic macro expansion. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, 1987.

[Mil87] Robin Milner. Dialogue with a proof system. In H. Ehrig, R. Kowalski, G. Levi, and U. Montarari, editors, *TAPSOFT '87*, volume 249 of *Lecture Notes in Computer Science*, pages 271–275, Berlin, 1987. Springer-Verlag.

[NPS86] Bengt Nordström, Kent Petersson, and Jan Smith. *An introduction to Martin-Löf's Type Theory (summmer 1986 draft)*. Department of Computer Sciences, University of Göteberg, Chalmers University of Technology, S-412 96 Goteberg, Sweeden, 1986. in preperation.

[Pau85] L.C. Paulson. Interactive theorem proving with cambridge lcf : A user's manual. Technical Report Report 80, Computer Lab., University of Cambridge, 1985.

[Pet82] Kent Petersson. A programming system for type theory. Technical Report 21, Programming Methodology Group, University of Goteborg/Chalmers Institute of Technology, 1982.

[Pot78] Garrel Pottinger. proofs of the normalization and church-rosser theorems for a typed λ-calculus. *Notre Dame Journal of Formal Logic*, 19(3):445–451, 1978.

[Pot81] Garrel Pottinger. The church-rosser theorem for the typed λ-calculus with surjective pairing. *Notre Dame Journal of Formal Logic*, 22(3):264–268, 1981.

[Qui53] Willard Quine. *From a Logical point of view*. Harper and Row, 1953.

[Rey81] J. C. Reynolds. The essence of algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North-Holland, 1981.

[Tro86] A. S. Troelstra. Strong normalization for typed terms with surjective pairing. *Notre Dame Journal of Logic*, 27(4):547–550, 1986.

[vD80] Diedrik van Daalen. The language theory of automath. Technical report, Technical University of Eindhoven, Eindhoven, The Netherlands, 1980. PhD thesis.

[WR10] Alfred Whitehead and Bertrand Russell. *Principia Mathematica.* Cambridge University Press, 1910.