Formalising Strong Normalisation Proofs of The Explicit Substitution Calculi in ALF[1]

Qiao Haiyan

Department of Computing Science
Glasgow University

A thesis submitted for the degree of Master of Science

December 1998

[1]This research was supported by EPSRC grant GR/L36963.

11438 (copy 1)

ACKNOWLEDGEMENT

I am grateful to my supervisor Fairouz Kamareddine and Joe Wells for countless discussions and encouragement. I want to thank Robert Pollack for many discussions, improving many proof techniques, careful proofreading and comments. I want to thank Michael Hedberg for many discussions. Thanks go to Rene Vestergaard and Richard Reid for proofreading and comments. Thanks go to Martin Hofmann for his nice comments. I want to thank the Department of Computing Science at Glasgow University for providing me a stimulating and friendly working environment. I wish to express my thanks for the EPSRC grant GR/L36963.

Summary

Explicit substitution calculi have become very fashionable in the last decade. The reason is that substitution calculi bridge theory and implementation and enable control over evaluation steps and strategies. Of the most important questions of explicit substitution calculi is that of the termination of the underlying calculus of substitution. For this reason, one finds with every new calculus of explicit substitution, a section devoted to the termination of substitutions. Those proofs of termination fall under two categories. Proofs that are easy because a decreasing measure can be established and proofs that are difficult because such a decreasing measure is not easy to establish.

Another fashionable subject has been the checking of proofs using a proof checker. This is useful because some proofs can be intricate and hard to believe if they are not proof checked.

This thesis investigates the methods to prove termination of explicit substitution calculi and their formalisations in the proof checker ALF. Two styles of explicit substitution calculi are chosen for this purpose, one is the calculus s whose termination is guaranteed by a decreasing weight, the other is the calculus σ whose termination is extremely complex. Two new termination proofs of the calculus s are given. All termination proofs of both s and σ presented in the thesis are formalised in ALF. During our process of formalisations we comment on what is needed to make a proof checkable during the checking process.

Contents

Chapter 1

Introduction

1.1 What is a calculus with explicit substitutions

Substitution is a very common operation in mathematics. There are substitutions where there are variables. *"Despite the fact that substitution is a "straightforward idea", it turns out to be surprisingly complicated to give a rigorous mathematical definition of the substitution process...Indeed, there is a long history of erroneous definition of "substitutions" in the literature of logic and programming semantics"* (Abelson and Sussman in [2]).

λ-calculus, invented by Church, has successfully provided the representation of computations and reasoning due to its simplicity and expressiveness. This has resulted in functional programming languages and systems of computer mathematics.

The syntax of the λ-calculus is very simple. Terms in λ-calculus are defined as:

$$\Lambda ::= V \mid \lambda V.\Lambda \mid \Lambda\Lambda$$

where V is a set of variables.

The main mechanism of λ-calculus is β-reduction (or β rule) which is usually defined as:

$$(\lambda x.t)s \rightsquigarrow t[x := s]$$

where $[x := s]$ (or $[s/x]$ in literature) is the substitution of the term s by the variable x.

In classical λ-calculus [4] there is no constructor for the expression $t[x := s]$ on the right hand side of the β rule, and the mechanism of substitution is usually left undefined or described at a meta-level by a specific and external formalism. It is in the process of implementation of the λ-calculus that the substitution is defined in small steps.

In λ-calculus with explicit substitutions the term $t[x := s]$ will become a legal term, a *closure* in terminology in functional programming, and there are both the β-rule and a description of the evaluation of the substitution, i.e. some rules describing how to evaluate the substitutions. Hence explicit substitutions mean here to make substitutions as concrete operations and as a part of the calculus, not as a meta-operation. In fact we must make the substitutions explicit when proving many basic properties of the λ-calculus, e.g. substitution lemma. Thus there is a conceptual gap between the theory of the λ-calculus and its implementation in programming languages and proof assistants.

Explicit substitution calculus was proposed to overcome the gap of the β-reduction $(\lambda x.t)s \rightsquigarrow t[x := s]$ and the implementation of the λ-calculus. In 1978 de Bruijn presented in [12] the first calculus of explicit substitutions which he called $C\lambda\xi\phi$ [23]. The calculus $\lambda\sigma$ proposed by Abadi et al. [1] is the most referred explicit substitution calculus. Several explicit substitution calculi have been proposed in the last fifteen years. For an overview see [21] and [23]. In section 1.3 we shall give a brief overview.

1.2 Why explicit substitutions

By representing the substitutions in the structure of terms and by providing (first order) reductions to propagate the substitutions, explicit substitution provides a number of benefits [19]. A major benefit is that explicit substitution allows much flexibility in the order of evaluation of a term. This is due to practical consideration, substitutions happen in a more controlled way [1]. Propagating substitutions through a particular subterm can wait until the subterm is the focus of computation. Obtaining more control over the ordering of evaluation has become an important issue in functional programming language implementation [17]. Another benefit is that explicit substitution allows formal modelling of the techniques used in real implementations, e.g., environments. Because explicit substitution is closer to real implementations, it has the potential to provide a more accurate cost model.

Proof assistants may benefits from explicit substitution, due to the desire to perform substitutions locally and in a formal manner. Local substitutions are needed as follows. Given $xx[x := y]$, one may not be interested in having yy as the result of $xx[x := y]$ but rather only $yx[x := y]$. In other words, one only substitutes one occurrence of x by y and continues the substitution later. Theorem provers like Nuprl and HOL implement substitutions which allows the local replacement of some abbreviated term. This avoids a size explosion when it is necessary to replace a variable by a huge term only in specific places to prove a certain theorem.

A formal theory of explicit substitution calculi helps in studying the termination and confluence properties of systems. Without formalisation, important properties such as the correctness of substitutions often remain unestablished, causing mistrust in the implementation. In fact, it is known that the first implementation of substitution in Automath was incorrect, and that most of the bugs in the implementation of LCF came from clashes of bound variables in strange situations [30]. As the implementation of substitution in many theorem provers is not based on a formal system, it is not clear what properties their underlying substitution has, nor can their implementations be compared. Thus, it helps to have a choice of explicit substitution systems whose properties have already been established. This is witnessed by the recent theorem prover ALF, which is formally based on Martin-Löf's type theory with explicit substitution [24].

1.3 Survey of explicit substitutions

Various calculi with explicit substitutions have been proposed in the last fifteen years. Amongst these we mention $C\lambda\xi\phi$ [12], λSUBST [10], $\lambda\sigma$ [1], $\lambda\sigma_{\Uparrow}$ [16], $\lambda\sigma_{SP}$ [32], $\lambda\upsilon$ [5], λs [20], λs_e [22] and $\lambda\zeta$ [28]. Every calculus has its operators to denote substitutions and rules to promote substitutions to variables. It is expected that explicit substitution calculi preserve all the properties of λ-calculus, e.g. strong normalisation and confluence. But it was found that an explicit substitution may not terminate, see [27]. How about the termination of the explicit substitution part without β-reduction?

It turns out that it is not at all trivial to prove the termination for explicit calculi. One of the reasons is that an explicit substitution calculus usually has many rules, which make the proof complicate and cumbersome.

When considering explicit substitutions, one comes to the variable renaming to prevent variable clashes. One way to solve this problem is to use de Bruijn indices. Then one actually makes renaming in a formal way. Bruijn's indices use natural numbers, the indices, as variable names. The index of a variable is the number of λ's one crosses before the λ that binds that variable. For instance in $\lambda x \lambda y \lambda z.x$ the index of the only occurrence of x is 3 and in the notation of λ-terms with indices, x will be replaced by 3. The indices allow us to associate directly a variable with its binder, therefore there is no need for the name of a variable next to each λ. For instance, $\lambda 1$ is equivalent to $\lambda x.x$, $\lambda 1(\lambda 12)$ is equivalent to $\lambda x.x(\lambda y.yx)$ and $\lambda\lambda\lambda 3$ is equivalent to $\lambda x \lambda y \lambda z.x$. Most of the explicit substitution calculi use de Bruijn's indices. Another way to deal with variable name clashes was proposed by J. McKinna and R. Pollack [26].

Basically there are two approaches to explicit substitutions. One is the $\lambda\sigma$ family, which including $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\sigma_{SP}$ and λv. The main innovation of the $\lambda\sigma$ is the division of terms in two sorts: sort: **term** and sort: **substitution**. λs and λs_e depart from $\lambda\sigma$ style in two ways. First, they keep the classical and unique sort *term* of the λ-calculus. Second, they do not use some of the categorical operators, especially those which are not present in the classical λ-calculus.

Suppose that we use λ*subst* denote a λ-calculus with explicit substitution, ***subst*** is the underlined calculus (without β-reduction rule). There are several properties of an calculus with explicit substitutions that are of interest:

1. Strong Normalisation (Termination) – the calculus ***subst*** terminate, i.e. no infinite derivations are possible;

2. Church-Rosser (Confluence) – the calculus λ***subst*** is confluent, i.e. the result of a computation does not depend on the computation path, on:

   (a) Ground terms (i.e. terms in λ calculus with explicit substitutions);

   (b) Open terms (i.e. terms with meta-variables).

3. Simulation of β-reduction – If a term s evaluates to t in the λ-calculus (using β-reduction), then s evaluates to t in $\lambda subst$ (using the β-rule and other substitution rules).

4. Preserving Strong Normalisation – terms which are strongly normalising in λ-calculus are strongly normalising in the calculus $\lambda subst$.

The calculus $\lambda\sigma$ did enjoy 1, 2a and 3. It did not process 2b however. Therefore $\lambda\sigma_{\Uparrow}$ was proposed. $\lambda\sigma_{\Uparrow}$ is a variant of $\lambda\sigma$ that satisfies 1, 2 and 3. Nevertheless, 4 remained unknown for $\lambda\sigma$ or $\lambda\sigma_{\Uparrow}$ until Melliès proved that $\lambda\sigma_{\Uparrow}$ (as well as both the rest of the $\lambda\sigma$ family and the categorical combinators) does not preserve SN [27]. Since then, it has remained an open problem whether there is indeed a calculus of substitutions that satisfies all the properties $1 \ldots 4$.

The calculus $\lambda\upsilon$ satisfies 1, 2a, 3 and 4. The calculus $\lambda\zeta$ satisfies 1, 2 and 4. Unfortunately, $\lambda\zeta$ is not able to simulate one step β-reduction. Instead, it simulates only a "big step" β-reduction [28].

Another line of explicit substitutions has been made in [18, 20, 22]. The calculus λs satisfies 1, 2a, 3 and 4. λs_e, an extension of λs, satisfies 1, 2 and 3. Kamareddine conjectured that λs_e preserves strong normalisation. Unfortunately, Bruno Guillome gave a counter example that a typed term is not terminating in λs_e. Therefore, it still remains a open problem to find an explicit substitution satisfying properties 1 $\ldots 4$. For an overview see [21].

1.4 Why formalising properties of explicit substitutions

Formalising mathematics and proof checking has been a fashionable subject, e.g. see [7, 3, 26] and [33]. It is hardly convincing if a complex proof has not been checked in a proof checker. As most calculi with explicit substitutions have complex sets of rules and proving their properties of interests are intricate, it is also necessary to formalise these calculi and check the proofs in some proof checker. Many implementations of type theory and programming languages need to deal with explicit substitutions, hence a formal theory of explicit substitutions is needed. So far we have not found an ideal calculus of explicit substitutions. For each explicit substitution calculus the proofs of termination and other properties have to be repeated. Through formalisation of explicit substitutions, we can have a better understanding of the properties of a calculus of explicit substitutions and the ways to prove them. Finally we hope to find a general way to prove termination and confluence of calculi of explicit substitutions.

In this thesis we are going investigate the strong normalisation process and ways to prove termination. Our final objective is to find a general method for proving termination of explicit substitutions and build a package of tools to help researchers to analyse any new explicit substitution calculus. To this end we first formalise normalisation proofs of some well studied explicit substitution calculi in ALF, a recent developed proof assistant based on Martin-Löf's type theory.

The importance of the proposed work has been recognised recently. In [33] Saïbi formalised the termination of the explicit substitution $\sigma_{\Uparrow}$ and the confluence of the calculus $\lambda\sigma_{\Uparrow}$. The termination of the calculus $\sigma_{\Uparrow}$ is not intricate because it uses a decreasing weight. While the termination proof of the calculus σ in [11] is difficult and intricate. Such a proof need to be checked to guarantee its correctness and to obtain a better understanding of the way to prove termination.

In summary, our work has the following objectives and advantages:

- Formally checking existing proofs of important properties of explicit substitutions (such as termination and confluence) in order to guarantee the correctness of these intricate proofs. This is particularly important because programming languages and theorem provers use explicit substitutions and hence it becomes vital that termination is guaranteed.

- Simplifying existing intricate proofs while guaranteeing general unified methods that can be used for a wide class of calculi. It is the hope that through formalisation, the difficulty can be eased and future proofs can benefit greatly.

- Reaching general proof environments for studying properties of explicit substitutions so that new calculi can be fed into this environments and their properties can be checked. This is important due to the active interests in studying and inventing new calculi of explicit substitutions.

1.5 This thesis

In this thesis we shall formalise strong normalisation proofs of well studied explicit substitution calculi σ [1] and s [20].

$\lambda\sigma$ is the first well studied calculus with explicit substitutions. The strong normalisation of σ-calculus was proved originally by the strong normalisation of $SUBST$ [15] and a translation from σ to $SUBST$. There are several strong normalisation proofs of σ, see [15, 11] and [35]. We have chosen to formalise the termination proof σ in [11] because it uses an interesting induction argument, although the proof is very complicated and intricate. It is interesting if this argument can be adapted to other calculi [11].

The calculus λs is quite a natural and simple calculus proposed by F. Kamareddine and A. Ríos, which is of different style from $\lambda\sigma$. The strong normalisation proof of the underlying calculus s is also simple and s enjoy many nice properties and has an extension which is confluent on open terms. The strong normalisation of s-calculus was proved by a strict translation from s to σ in [20]. We shall give two other strong normalisation proofs of the calculus s, one is by a decreasing weight and the other is by induction in this thesis, both of them are much simpler than the original proof in [20]. These strong normalisation proofs are also formalised in ALF in this thesis.

- In chapter 2 we introduce some basic notions about abstract reduction systems.
- In chapter 3 we recall the calculi $\lambda\sigma$ and λs, and give two new strong normalisation proofs of the calculus s.
- In chapter 4 we give a brief introduction to ALF.
- In chapter 5 we formalise some basic notions of abstract rewriting systems in ALF.
- In chapter 6 the calculus $\lambda\sigma$ is implemented in ALF.
- A context calculus for σ_0, a variant of σ, is formalised in Chapter 7, including all the notions and proofs presented in [11].
- The strong normalisation proof of the calculus σ is checked in chapter 8.

- In chapter 9 we check two strong normalisation proofs of s in ALF, one is given in chapter 3 and the other was given in [20].
- In chapter 10 we conclude.
- For details on all the proofs we refer the reader to http://www.dcs.gla.ac.uk/people/personal/qiao/papers/sig0sn.ps.

Chapter 2

Abstract Reduction Systems

In this chapter we define some properties of *abstract reduction systems* and state some simple facts about them. This is done only in as far we need those definitions and facts in the sequel as an explicit substitution calculus is a special *abstract reduction system.*

2.1 Abstract Reduction Systems

Definition 2.1.1 *An* abstract reduction system *(ARS) is a structure $\mathcal{A} = (A, \{\rightarrow_\alpha \mid \alpha \in I\})$ consisting of a set A and a set of binary relations $\rightarrow_\alpha$ on A, indexed by a set I. For $\alpha \in I$, the relations $\rightarrow_\alpha$ are called* reduction *or* rewrite *relations. In the case of just one reduction relation, we simply write $\rightarrow$.*

Let $\mathcal{A} = (A, \{\rightarrow_\alpha \mid \alpha \in I\})$ be an ARS and let $\alpha \in I$. If for $a, b \in A$ we have $(a, b) \in \rightarrow_\alpha$, we write $a \rightarrow_\alpha b$ and call b a *one-step* (α) reduct of a. A *reduction sequence* with respect to $\rightarrow_\alpha$ is a (finite or infinite) sequence $a_0 \rightarrow_\alpha a_1 \rightarrow_\alpha a_2 \rightarrow_\alpha \cdots$. If a finite reduction sequence ends in b, then it is called a reduction sequence *from a to b*. The element b is called an (α)*reduct* of a. Reduction sequences are also called *reduction paths.* A *reduction step* is a specific occurrence of $\rightarrow_\alpha$ in a reduction sequence. A *reduction step from a to b* is a specific occurrence of $a \rightarrow_\alpha b$. The *length* of a finite reduction sequence is the number of reduction steps occurring in this reduction sequence.

The inverse relation of $\to_\alpha$ is $\to_\alpha^{-1}$, also denoted by $\leftarrow_\alpha$. The transitive closure of $\to_\alpha$ is written as $\to_\alpha^+$, and the transitive reflexive closure of $\to_\alpha$ is $\to_\alpha^*$ or $\twoheadrightarrow$.

We shall consider ARS with only one reduction relation later on.

Definition 2.1.2 *(Confluence) Let $\mathcal{A} = (A, \to)$ be an ARS.*

- *$a \in A$ is* weekly confluent *if $\forall b, c \in A \exists d \in A (c \leftarrow a \to b \Rightarrow c \twoheadrightarrow d \twoheadleftarrow b)$.*

  The reduction relation $\to$ is weakly confluent *or* weakly Church-Rosser *(WCR) if every $a \in A$ is weakly confluent*

- *$a \in A$ is* confluent *if $\forall b, c \in A \exists d \in A (c \twoheadleftarrow a \twoheadrightarrow c \Rightarrow c \twoheadrightarrow d \twoheadleftarrow b)$. The reduction relation $\to$ is* confluent *or* Church-Rosser *(CR), if every $a \in A$ is confluent.*

Definition 2.1.3 *(Normalisation) Let $\mathcal{A} = (A, \to)$ be an ARS.*

- *$a \in A$ is a* normal form *if there exists no $b \in A$ such that $a \to b$.*

- *$a \in A$ is* weakly normalising *(WN) if $a \twoheadrightarrow b$ for some normal form $b \in A$. The reduction relation $\to$ is* weakly normalising *if every $a \in A$ is weakly normalising.*

- *$a \in A$ is* strongly normalising *(SN) or* Noetherian *if every reduction sequence starting from a is finite. The reduction relation $\to$ is* strongly normalising *if every $a \in A$ is strongly normalising.*

A partial ordered set $(A, \prec)$ is *well founded* (WF) if there exists no infinite descending chain $\cdots \prec a_2 \prec a_1 \prec a_0$. We shall say the set A or the relation $\prec$ is well founded where we mean $(A, \prec)$ is well founded.

Obviously, SN implies WN and $\to$ is SN if and only if $\leftarrow$ is WF.

SN and CR are two important properties of ARS, which are expected to be satisfied for some ARS', especially for calculi of explicit substitutions. SN means every computation (reduction sequence) starting from a term will terminate at a result, and CR means that the result of a computation of a term is independent of the computation strategy (reduction path). Newman's lemma describes the relation between SN and CR.

Lemma 2.1.4 *(Newman's lemma) For every ARS we have SN $\wedge$ WCR $\Rightarrow$ CR.*

For the proof see [4], [6].

2.2 Termination

Terminating systems are variously called *strongly normalising, finitely terminating and Noetherian.*

Basically there are two ways to prove termination of an abstract reduction system(ARS): *syntactical* methods and *semantical* methods.

In a syntactical method, terms are ordered by a careful analysis of the term structure, such that a term is always greater that its proper sub-terms. A well-known representative of this method is the *recursive path order*, see e.g. Dershowitz [13](1987).

In a semantical method terms are interpreted in some well-known well-founded ordered set in such a way that each rewrite chain will map to a descending chain, and hence will terminate. Most semantical methods have focussed on choosing the natural numbers as the well-founded order set. The method of ***polynomial interpretations*** as given by Lankford (1979), Ben-Cherifa and Lescanne (1987) can be seen as a particular case of a semantical method on natural numbers. Zantema introduced the notion of a *monotone algebra* [35] as the natural concept for semantical methods, and proved:

Proposition 2.2.1 *(Zantema) An ARS is terminating if and only if it admits a compatible non-empty well-founded monotone algebra.*

An explicit substitution calculus is usually defined inductively, with variables as base terms and some operations as constructors. Therefore we can define the length of the terms. The reduction rules are inductively defined basic rules plus compatibles rules for those operations. For any term s, there exist finite possible one step reducts of s. If s is strongly normalising, then every reduction sequence from s is finite, hence by König's Lemma there exists the longest reduction path starting at s . Therefore we can have the notion of $length(s)$ when s is strongly normalising.

Termination of an explicit substitution calculus is proved usually by the interpretation method [23] or by finding an induction argument, which is usually induction on lexicographic order [11].

The interpretation method relies on the naive idea that for proving termination of reduction systems it is natural to associate a natural number $[\![t]\!]$ with each ground term t and to prove that reduction always decreases this number. In the case terms containing meta-variables, a term $t(x_1, \cdots, x_n)$ is associated with a function over the naturals $[\![t(x_1,\cdots,x_n)]\!](X_1,\cdots,X_n)$ that we call an interpretation. Interpretations are extended to term from interpretations given for basic operators. Proving that a reduction system (l_i,r_i) terminates boils down proving that the function $[\![l_i]\!]$ bounds the function $[\![r_i]\!]$, i.e., for all its values. Most of the interpretations are restricted to polynomials or polynomials and exponentials (elementary functions).

We shall often prove termination by finding a ***strict translation*** from the calculus to some ARS which is terminating, e.g. [11], [20],

Definition 2.2.2 *Let $(C_1,\rightarrow_1),(C_2,\rightarrow_2)$ be two ARS. We call* strict *those interpretations $f: C_1 \rightarrow C_2$ such that if $a \rightarrow_1 b$ then $f(a) \rightarrow_2^+ f(b)$.*

It is obvious that termination of C_2 and the existence of a strict interpretation of C_1 into C_2 yield termination of C_1.

Proposition 2.2.3 *Suppose $(C_1,\rightarrow_1),(C_2,\rightarrow_2)$ are two ARS. If there exists a strict interpretation from $(C_1,\rightarrow_1)$ to $(C_2,\rightarrow_2)$ and $(C_2,\rightarrow_2)$ is terminating, then $(C_1,\rightarrow_1)$ is also terminating.*

The termination of the calculus s , the calculus v [5] and the calculus $\sigma_{\Uparrow}$ [16] can be obtained by the interpretation method [23]. But it seems that this method can not be applied to σ. The termination of the s calculus can be proved by a strict translation from s to σ [20], and also by induction on a tuple of depth and length (see section 3.2). The termination of σ was proved by a strict interpretation of σ to σ_0, while the termination of σ_0 was proved by induction on some tuple of depth and length [11], see also chapters 7 and 8 in this thesis.

Chapter 3

Two Calculi of Explicit Substitutions

In this chapter we present the definitions of the two calculi of explicit substitutions: $\lambda\sigma$ and λs. We also provide two new strong normalisation proofs of the calculus s in section 3.2.

3.1 The Calculus $\lambda\sigma$

$\lambda\sigma$-calculus [1] is a refinement of the λ-calculus where substitutions are manipulated explicitly. $\lambda\sigma$ provides a setting for studying the theory of substitutions, with pleasant mathematical properties. Moreover, it is strongly connected with the categorical understanding of the λ-calculus, where a substitution is interpreted as a composition [10]. In this section we give the formal definitions of the $\lambda\sigma$-calculus and some of its properties.

3.1.1 Definition of the calculus $\lambda\sigma$

As we said, in explicit substitution calculi, substitutions are delayed and explicitly recorded; the application of substitutions is independent, and not coupled with the β-rule.

Substitutions have syntactic representations, and if a is a term and s is a substitution then the term $a[s]$, which is called a closure, represents a with the substitution s. We now express a β rule with delayed substitution, called Beta:

$$(\lambda x.a)b \to_{Beta} a[(b/x) \cdot id]$$

where $(b/x) \cdot id$ is the syntax for the substitution that replace x with b and affects no other variables ("$\cdot$" represents extension and id the identity substitution).

Definition 3.1.1 *The syntax of $\lambda\sigma$-calculus is given by:*

Let a, b range over $\Lambda\sigma^t$, the set of terms, s, t range over $\Lambda\sigma^s$, the set of substitutions.

Terms $a, b ::= 1 \mid ab \mid \lambda a \mid a[s]$

Substitutions $s, t ::= id \mid\uparrow\mid a \cdot t \mid s \circ t$

The set, denoted σ, of the rules which propagate the substitutions is the following:

$(VrId)$	$1[id] \to 1$	(IdL)	$id \circ s \to s$
$(VrCons)$	$1[a \cdot s] \to a$	$(ShId)$	$\uparrow \circ id \to \uparrow$
(App)	$(ab)[s] \to (a[s])(b[s])$	$(ShCons)$	$\uparrow \circ (a \cdot s) \to s$
(Abs)	$(\lambda a)[s] \to \lambda(a[1 \cdot (s \circ \uparrow)])$	(Map)	$(a \cdot s) \circ t \to a[t] \cdot (s \circ t)$
$(Clos)$	$(a[s])[t] \to a[s \circ t]$	(Ass)	$(s_1 \circ s_2) \circ s_3 \to s_1 \circ (s_2 \circ s_3)$

The $\lambda\sigma$-calculus is the union of the σ rules with the following *Beta* rule:

$$(\lambda a)b \to a[b \cdot id]$$

The *Beta* rule eliminates λ's and creates substitutions; the function of the σ rules is to eliminate substitutions.

If s represents the infinite substitution $\{a_1/1, a_2/2, a_3/3, \cdots\}$, then the syntax of substitutions can be described intuitively:

- id is the identity substitution $\{i/i\}$ (for all i);
- $\uparrow$ is the substitution $\{(i+1)/i\}$; for example, $1[\uparrow] = 2$. Thus, $n+1$ can be encoded as $1[\uparrow^n]$, where $\uparrow^n$ is the composition of n shifts: $\uparrow \circ \cdots \circ \uparrow$.
- $i[s]$ is the value of the De Bruijn index i in the substitution s, also written $s(i)$ when s is viewed as a function;

- $a \cdot s$ is the substitution $\{a/1, s(i)/(i+1)\}$; for example,

$$a \cdot id = \{a/1, 1/2, 2/3, \cdots\}$$

$$1 \cdot \uparrow = \{1/1, \uparrow(1)/2, \uparrow(2)/3, \cdots\} = id$$

- $s \circ t$ (the composition of s and t) is the substitution such that $a[s \circ t] = a[s][t]$, hence $s \circ t = \{s(i)/i\} \circ t = \{s(i)[t]/i\}$ and for example,

$$id \circ t = \{id(i)[t]/i\} = \{t(i)/i\} = t$$

$$\uparrow \circ (a \cdot s) = \{\uparrow(i)[a \cdot s]/i\} = \{s(i)/i\} = s$$

3.1.2 σ_0: a variant of σ

The strong normalisation of σ is proved in [11] by the strong normalisation of σ_0, an economic variant of σ, and a strict translation from σ to σ_0 in [11]. σ_0 is one sort calculus which treats both $\circ$ and $[]$ as $\circ$, observing that $\circ$ and $[]$ behave in the same way. Now we give the definition of σ_0:

Definition 3.1.2 *The syntax of the σ_0-calculus is given by:*

Terms of $\Lambda\sigma_0$ $\quad s, t ::= 1 \mid id \mid \uparrow \mid \lambda s \mid s \circ t \mid s \cdot t$

The set, denoted σ_0, of rules of the calculus is the following:

$(VrId)$	$1 \circ id \to 1$	$(ShId)$	$\uparrow \circ id \to \uparrow$
$(VrCns)$	$1 \circ (s \cdot t) \to s$	$(ShCons)$	$\uparrow \circ (s \cdot t) \to t$
(Abs)	$(\lambda s) \circ t \to \lambda(s \circ (1 \cdot (t \circ \uparrow)))$	(Map)	$(s \cdot t) \circ u \to (s \circ u) \cdot (t \circ u)$
(IdL)	$id \circ s \to s$	(Ass)	$(s \circ t) \circ u \to s \circ (t \circ u)$

Proposition 3.1.3 *The calculus σ_0 is strongly normalising.*

The termination of σ_0 was proved in [11] by introducing a context calculus. The whole proof is very complicated, so it is not given here. We shall implement the context calculus and the whole strong normalisation proof in ALF in chapter 7 and chapter 8.

3.1.3 Strong normalisation of the calculus σ

There are various strong normalisation proof for σ. The first strong normalisation proof of σ is based on the strong normalisation of $SUBST$ [15], which is, within CCL, the set of rewriting rules that compute the substitution. New proofs were give by P.-L.Curien [11] and Zantema [35].

In [11] the termination of σ was proved by a strict translation from σ to σ_0 and the termination of σ_0. This is the termination proof of σ that we will formalise.

The interpretation function from $\Lambda\sigma$ to $\Lambda\sigma_0$ is given by the following definition:

Definition 3.1.4 *Let $F : \Lambda\sigma \to \Lambda\sigma_0$ be the following interpretation:*

$$\begin{array}{ll} F(1) = 1 & F(id) = id \\ F(ab) = F(a)F(b) & F(\uparrow) = \uparrow \\ F(\lambda a) = \lambda(F(a)) & F(a \cdot s) = F(a) \cdot F(s) \\ F(a[s]) = F(a) \circ F(s) & F(s \circ t) = F(s) \circ F(t) \end{array}$$

Then we can prove the following:

Lemma 3.1.5 *If $a \to_\sigma b$ then $F(a) \to_{\sigma_0} F(b)$*

This is checked easily in ALF.

We state two properties of $\lambda\sigma$ without giving their proofs:

Theorem 3.1.6 *The calculus σ is confluent.*

This can be proved by Newman's lemma.

Theorem 3.1.7 *The calculus $\lambda\sigma$ $(Beta + \sigma)$ is confluent.*

The proof relies on the termination and confluence of σ, the confluence of the classical λ-calculus, and Hardin's interpretation technique. For a proof, see [1].

3.2 The Calculus λs

The calculus λs is a λ-calculus with explicit substitutions, which was proposed by F. Kamareddine and A. Ríos in quite a natural way [20].

We shall introduce the calculus λs in this section and give three strong normalisation proofs of the calculus s. The first one in section 3.2.2 was presented in [20] and the other two are new. The three proofs use three different methods, which are the ways to prove termination of explicit substitutions at present.

3.2.1 Definition of the calculus λs

Definition 3.2.1 *The set of terms of the λs-calculus, noted Λs is given as follows:*
Terms of Λs $\quad a, b ::= \mid N \mid ab \mid \lambda a \mid a\sigma^i b \mid \varphi^i_k a$

where $i \geq 1, k \geq 0$.

Definition 3.2.2 *The λs-calculus is given by the following rules:*

$$\begin{array}{ll}
\sigma - generation & (\lambda a)b \to a\sigma^1 b \\
\sigma - \lambda - transition & (\lambda a)\sigma^i b \to \lambda(a\sigma^{i+1} b) \\
\sigma - app - transition & (a_1 a_2)\sigma^i b \to (a_1\sigma^i b)(a_2\sigma^i b) \\
\sigma - destruction & n\sigma^i b \to \begin{cases} n-1 & if\, n > i \\ \varphi^i_0 b & if\, n = i \\ n & if\, n < i \end{cases} \\
\varphi - \lambda - transition & \varphi^i_k(\lambda a) \to \lambda(\varphi^i_{k+1} a) \\
\varphi - app - transition & \varphi^i_k(a_1 a_2) \to (\varphi^i_k a_1)(\varphi^i_k a_2) \\
\varphi - destruction & \varphi^i_k n \to \begin{cases} n+i-1 & if\, n > k \\ n & if\, n \leq k \end{cases}
\end{array}$$

We use λs to denote this set of rules. The calculus of substitutions associated with the λs-calculus is the rewriting system whose rules are λs - $\{\sigma$-generation$\}$ and we call it s-calculus.

This calculus has only one sort and no composition. In λs a clousre is denoted as $a\sigma^i b$, where only the occurences of one index i is replaced by term b in term a. Compare with $\lambda\sigma$ where occurences of many indices can be replaced simutaneously. $\varphi^i_k a$ is a family of renaming functions.

Here is the theorem that we will formalise in ALF in this thesis.

Theorem 3.2.3 *s-calculus is strongly normalizing.*

The strong normalisation of the calculus s was proved by a strict translation from s to σ (and the strong normalisation of the calculus σ) in [20]. We shall give the proof sketch in section 3.2.2. In section 3.2.3 we shall give another induction proof of this theorem. We shall formalise both these proofs in section 9.2 and 9.3.

Theorem 3.2.4 *s-calculus is local confluent.*

This can be checked directly.

Theorem 3.2.5 *s-calculus is confluent.*

This is proved by Newman's lemma.

Theorem 3.2.6 *The calculus λs is confluent.*

For a proof see [20].

3.2.2 Strong normalisation proof of s via σ

We show strong normalisation of s by giving a strict translation from s to σ, the proof was given in [20].

Before giving the translation, we first introduce some notations.

Definition 3.2.7 *For $k \geq 0$ and $i \geq 1$ we define $s_{ki} = 1 \cdot 2 \cdot \ldots \cdot k \cdot \uparrow^{k+i-1}$(we use the convention $s_{0i} = \uparrow^{i-1}$).*

Definition 3.2.8 *Let $b \in \Lambda\sigma^t$, we define a family of substitutions $(b_k)_{k\geq 1}$ as follows:*

$b1 = b[id] \cdot id$

$b_2 = 1 \cdot b[\uparrow] \cdot \uparrow$

$\ldots$

$b_{i+1} = 1 \cdot 2 \cdot \ldots \cdot i \cdot b[\uparrow^i] \cdot \uparrow^i$

Definition 3.2.9 *The translation function $T : \Lambda s \to \Lambda\sigma$ is defined by:*

$T(n) = n$

$T(ab) = T(a)[T(b)_i]$

$T(\lambda(a) = \lambda(T(a))$

$T(a\sigma^i b) = T(a)[T(b)_i]$

$T(\varphi^i_k a) = T(a)[s_{ki}]$

where $\Lambda\sigma = \Lambda\sigma^t \cup \Lambda\sigma^s$.

Theorem 3.2.10 *If $a \to_s b$ then $T(a) \to_\sigma^+ T(b)$*

We shall give an ALF proof of this theorem in Section 9.3. The strong normalization proof of σ will be given in chapter 8. Then we will get the strong normalization of the calclus s.

3.2.3 A direct strong normalisation proof of s

Now we give the direct proof that s is strongly normalising. This is done by structural induction, the method used to prove strong normalisation of σ_0, a variant of σ. But the proof is much easier for s.

Let SN be the set of all strongly normalising terms. For a strongly normalising term t, $dpth(t)$ is the length of the longest derivations[1], $lgth(t)$ is the number of variables and operations defined as follows:

$lgth(n) = 1$

$lgth(ab) = lgth(a) + lgth(b) + 1$

$lgth(\lambda(a)) = lgth(a) + 1$

$lgth(a\sigma^i b) = lgth(a) + lgth(b) + 1$

$lgth(\varphi_k^i a) = lgth(a) + 1$

Remark 3.2.11 *Let $a, b \in \Lambda s$.*

1. *If a,b $\in$ SN and $a \longrightarrow^+ b$, then dpth(a) $>$ dpth(b)*
2. *If a is a nontrivial sub-term of b, then lgth(a) $<$ lgth(b).*

Since there are no rules of the calculus s contain "λ" or "apply" as head symbol, hence we have:

Lemma 3.2.12 *Let $a, b \in \Lambda s$.*

1. *ab$\in$ SN if and only if $a \in$ SN and b$\in$ SN.*
2. *$\lambda a \in$ SN if and only if $a \in$ SN.*

Therefore, in order to prove that all terms are terminating, we need only to check that

if $a, b \in$ SN, then $\varphi_k^i a \in$ SN and $a\sigma^i b \in$ SN.

[1] $dpth(t)$ is well defined for strongly normalising term t by König Lemma.

Lemma 3.2.13 *If $a \in SN$, then $\varphi^i_k a \in SN$ for all $i \geq 1$, $k \geq 0$.*

Proof:

We prove the lemma by induction on (dpth(a), lgth(a)), where we use lexicographic order $(m, n) > (m', n') \iff m > m' \vee (m = m' \wedge n > n')$.

Suppose that a $\in$ SN and for any b$\in$ SN, which satisfying

(dpth(b),lgth(b)) < (dpth(a),lgth(a)), $\varphi^i_k b \in$ SN.

Now we prove that $\varphi^i_k a \in$ SN.

If $\varphi^i_k a$ is in normal form, the proposition is trivial. Let us consider a reduction starting at $\varphi^i_k a$. There are three cases:

1. $a \to a'$, hence $\varphi^i_k a \to \varphi^i_k a'$.

   Then we have $(dpth(a'), lgth(a') < (dpth(a), lgth(a))$. From the IH,

   $\varphi^i_k a' \in$ SN.

2. $\varphi^i_k a \to (\varphi^i_k a_1)(\varphi^i_k a_2)$ because a $= a_1 a_2$.

   In this case $dpth(a_1) \leq dpth(a)$, and $lgth(a_1) < lgth(a)$. Hence $(dpth(a_1), lgth(a_1)) < (dpth(a), lgth(a))$. Then IH can be applied and $\varphi^i_k a_1$ $\in$SN. Similarly, $\varphi^i_k a_2 \in$ SN by IH, and $(\varphi^i_k a_1)(\varphi^i_k a_2) \in$ SN by lemma1.

3. $\varphi^i_k a \to \lambda \varphi^i_{k+1} a_1$ because of $a \equiv \lambda a_1$.

   We have $(dpth(a_1), lgth(a_1)) < (dpth(a), lgth(a))$. From the IH and Lemma1,$\lambda \varphi^i_{k+1} a_1 \in$ SN. This means $\varphi^i_k a$ can be only reduced to a strongly normalizing term, so $\lambda \varphi^i_k a$ itself is strongly normalising.

□

Lemma 3.2.14 *If $a, b \in SN$, then $a\sigma^i b \in SN$ for all $i \geq 1$, $k \geq 0$.*

Proof:

By induction on (dpth(a), lgth(a), dpth(b), lgth(b)) with the lexicographic order.

Suppose that a,b $\in$ SN, and for any terms a', b'which satisfy

$(dpth(a'), lgth(a'), dpth(b'), lgth(b')) < (dpth(a), lgth(a), dpth(b), lgth(b))$

we have

$a'\sigma^i b' \in SN$ for all $i \geq 1$, $k \geq 0$.

Let us consider a reduction beginning at $a\sigma^i b$. There are four cases:

1. $a \to a_1$, hence $a\sigma^i b \to a_1\sigma^i b$. Then we have

   $(dpth(a_1), lgth(a_1), dpth(b), lgth(b)) < (dpth(a), lgth(a), dpth(b), lgth(b))$

   because $dpth(a_1) < dpth(a)$. Therefore IH can be applied, and $a_1\sigma^i b \in$ SN.

2. $b \to b_1$, hence $a\sigma^i b \to a\sigma^i b_1$. This is similar to the case above.

3. $(a_1a_2)\sigma^i b \to (a_1\sigma^i b)(a_2\sigma^i b)$ because $a = a_1a_2$. We have

   $(dpth(a_1), lgth(a_1), dpth(b), lgth(b)) < (dpth(a), lgth(a), dpth(b), lgth(b))$

   because $(dpth(a_1), lgth(a_1) < (dpth(a), lgth(a))$. From the IH, we have $a_1\sigma^i b \in$ SN.

   Similarly $a_2\sigma^i b \in$ SN. Then from lemma 9.2.1 $(a_1\sigma^i b)(a_2\sigma^i b) \in$ SN.

4. $(\lambda a_1)\sigma^i b \to \lambda(a_1\sigma^{i+1}b)$ because $a = \lambda a_1$.

   In this case,

   $(dpth(a_1), lgth(a_1) < (dpth(a), lgth(a))$,

   therefore

   $(dpth(a_1), lgth(a_1), dpth(b), lgth(b)) < (dpth(a), lgth(a), dpth(b), lgth(b))$

   From the IH, $a_1\sigma^{i+1}b \in$ SN, and $\lambda(a_1\sigma^{i+1}b) \in$ SN from lemma1. Hence for all the cases, $a\sigma^i b$ can only be reduced to a strongly normalising term. So $a\sigma^i b \in$ SN.

This also completes the strong normalisation proof of s.

□

3.2.4 Interpretations for the termination of s

The interpretation method can be used to prove the termination of the calculus s. The weight of the term is defined as follows:

Definition 3.2.15 *The polynomial interpretations for s are defined by induction on* *the structure of the terms in Λs:*

$$\begin{aligned}
&[\![n]\!] = 2 \\
&[\![ab]\!] = [\![a]\!] + [\![b]\!] + 1 \\
&[\![\lambda a]\!] = [\![a]\!] + 1 \\
&[\![a\sigma^i b]\!] = [\![a]\!]([\![b]\!] + 1) \\
&[\![\varphi_k^i a]\!] = 2[\![a]\!]
\end{aligned}$$

Then we can prove the following theorem:

Theorem 3.2.16 *For any $a, b \in \Lambda s$, if $a \to_s b$ then $[\![a]\!] > [\![b]\!]$.*

This theorem gives another termination proof of s by proposition 2.2.3. This theorem was checked in ALF by some trivial inequalities including the following inequality:

For any $a \in \Lambda s$, $[\![a]\!] \geq 2$.

(which can be proved by induction on the structure of the terms of s).

Chapter 4

The Proof Assistant ALF

4.1 About Martin-Löf's Type Theory

The ALF (" Another Logical Framework") system is a proof assistant which supports proof in Martin-Löf's type theory. This theory was introduced by Martin-Löf in the beginning of the seventies [25] and exists in several versions. ALF implements the most recent version presented by Martin-Löf in 1986. This version is monomorphic and intensional, see Nordström, Petersson and Smith [29] for detailed description. ALF also support a rich class of inductive definitions, see Dybjer [14] and definition by pattern matching, see Coquand [8].

Martin-Löf's type theory was originally developed with the aim of being a clarification of constructive mathematics, but unlike most other formalisations of mathematics it is not based on first order predicate logic. Instead, predicate logic is interpreted within type theory through the correspondence between propositions and sets, the basic idea behind Martin-Löf's type theory, or the Curry-Howard interpretation of propositions as types (sets). A proposition is interpreted as a set whose elements represent the proofs of the proposition. Hence, a false proposition is interpreted as the empty set and a true proposition as a non-empty set. To prove a proposition is true is to prove the set is inhabited.

There are basically two ways of introducing types in Martin-Löf's type theory: function types and inductively defined sets. The function types make it possible to express rules in a natural deduction style and logic can then be introduced by the idea of proposition as sets. Because of the possibility of introducing sets by induction, type theory is an open theory; it is in this sense that the theory may serve as a logical framework.

For every inductively defined set, there are *one formation rule*, *introductions rules* and *one elimination rule.* The formation rule says how to form a set, the introduction rules say how to form the elements of the set, and the elimination rule says the induction principle for this set, i.e. how to prove all the elements of the set satisfy some property. Basically one states in the elimination rule if for every constructor one can show the property holds, then the property holds for all the elements of the set. Another way to look at the elimination rule is that it says there are no other objects in this set except those given by introduction rules. There is a general scheme to derive the elimination rule from the introduction rules of a set, see [14].

For example, the set of natural numbers Nat is formed by the *formation rule*:

$$\frac{}{Nat \in \mathbf{Set}}$$

the elements of the set Nat is defined by two introduction rules:

$$\frac{}{0 \in Nat}$$

$$\frac{a \in Nat}{s(a) \in Nat}$$

Here Nat is a set having two *constructors*: the nullary 0 and the unary s, which is a function from Nat to Nat.

The elimination rule is just the induction principle:

$$\frac{\begin{array}{l} C(v)set[v \in Nat] \\ a \in Nat \\ d \in C(0) \\ e(x, y) \in C(s(x))[x \in Nat, y \in C(x)] \end{array}}{natrec(a, d, e) \in C(a)}$$

In ALF the introduction rules of Nat look like:

$$Nat \in \mathbf{Set}$$

$$0 \in Nat$$

$$s \in (Nat)Nat$$

We will present rules in a natural deduction style or in ALF style above. We will use $a \in A$ or $a : A$ to denote a is an element (object) of the set (type) A.

A proposition is proved by constructing a proof object, or an element of the set in ALF.

Objects of a type are formed from constants and variables using application and abstraction: applying a function to an object:

$$\frac{c \in (x \in A)B \qquad a \in A}{c(a) \in B[x := a]}$$

forming a function by abstraction:

$$\frac{b \in B[x \in A]}{[x]b \in (x \in A)B}$$

4.2 About explicit substitution in Martin-Löf's Type Theory

In the original presentation of Martin-Löf's type theory, substitutions are left unspecified. Consider for instance a rule allowing the formation of a type α depending on variables. Such a rule should be justified by explaining what it is to simultaneously assign in α objects of appropriate types to the variables on which α depends, so that the result of this process is a type. A. Tasistro gave a presentation of Martin-Löf' type theory with explicit substitutions [34].

Tasistro's presentation contains the following forms of judgement:

$$\begin{array}{lllll} \Gamma \;\; \mathit{context} & \delta : \Gamma \to \Delta & \alpha \;\; \mathit{type}[\Gamma] & a : \alpha[\Gamma] & \beta : \alpha \to \mathit{type}[\Gamma] \\ \Gamma \preceq \Delta & \delta_1 = \delta_2 : \Gamma \to \Delta & \alpha = \beta \;\; \mathit{type}[\Gamma] & a = b : \alpha[\Gamma] & \beta_1 = \beta_2 : \alpha \to \mathit{type}[\Gamma] \end{array}$$

The rules are classified into three groups: general rules, rules for substitutions and rules for families of types and types. Expressions with substitutions are denoted as $e\delta$ where δ is a substitutions. For details of the calculus, see [34].

The main difference between Tasistro's approach and Abadi's is that in Tasistro's presentation, substitution does not commute with abstraction, i.e. it will never by pushed from without into the scope of a bound variable. Instead, the operation of a substitution on an abstraction shall be delayed or "explicitly recorded". As a consequence, there is no need of considering renaming in order to avoid capture of variables when combining substitutions and abstractions. Another difference is Abadi does not consider systems with families of types.

Explicit substitution calculus in Martin-Löf's type theory is also intended to be used to actually carry out mathematical proofs. In fact, ALF is a proof editor based on Martin-Löf's type theory with explicit substitutions.

By considering the activity of actual proof construction of edition it is led to introducing the notion of an "incomplete" or "scratch" construction. An incomplete construction contains symbols that stand for constructions yet to be performed. These symbols are called placeholder and denoted as $?_i$ in ALF, where i is an index. Explicit substitution is important to deal with incomplete proofs. For example, assume we want to reduce the term

$$([x]f(x, ?_1))a$$

where a is the argument to the function $[x]f(x, ?_1)$ which contains a placeholder $?_1$. Since $?_1$ is within the scope of the binder x, it may depend on x, that is its local context is $[x := A]$ for some type A. Without explicit substitutions, this term could not be reduced any further, since what would we do with the second argument, i.e. the placeholder? We cannot forget that once the placeholder $?_1$ is instantiated, say to x, then x should be replaced by a. With explicit substitution we can safely reduce the term to

$$f(a, ?_1\{x := a\})$$

and when the placeholder $?_1$is instantiated to x we have the term

$$f(a, x\{x := a\}) = f(a, a)$$

since the term a can simply be looked up in the substitution. The possibility of reducing terms as far as possible is important since it improves the unification algorithm. If we are interested in finding instantiations to the place-holders in the equation

$$([x]f(0, ?_1)a = f(?_2, a)$$

where we can reduce the left-hand side expression yielding

$$f(0, ?_1\{x := a\}) = f(?_2, a)$$

which can be simplified to

$$0 = ?_2$$

$$?_1\{x := a\} = a$$

Hence, the unification found an instantiation of the placeholder $?_2$ which would not have been possible without explicit substitutions.

4.3 The proof assistant ALF

ALF is an interactive proof assistant, and the proof is processed on the screen directly by the user and finished step by step. This also means that it must be possible to deal with incomplete proofs, i.e. proof objects which represent incomplete proofs. In ALF place-holders are used to represent those parts of objects which are to be filled in. The expression

$$? \in A$$

expresses a state of an ongoing process of finding an object in the type A.

There are four ways of refining a placeholder:

- The placeholder is replaced by a constant c. This is correct if the type of c is equal to A.
- The placeholder is replaced by a variable x, where x must be in the local scope of the placeholder.
- The placeholder is replaced by an abstraction $[x]? \in A$ if A is equal to function type $(y \in B)C$. We are constructing a solution to the problem C under the assumption that we have a solution to B.
- Finally, the placeholder can be replaced by an application $c(?_1, \ldots, ?_n)$. In this case we can divide the problem to several subproblems.

ALF implements a *monomorphic* version of type theory, this means that all type information is in the term. As a consequence it maintains a lot of type information which is redundant or uninteresting, and the full size of the proofs can be very large. However, the user can instruct ALF to suppress unwanted type information during display. This also makes the proofs nicer and more readable.

We have used Window ALF, a version of ALF which was implemented by Lena Magnusson [24]. Because of the Curry-Howard analogy, to prove a theorem in ALF is the same as writing a program "witnessing" the truth of the theorem. This is a fundamental difference between ALF and HOL (and many other proof-assistants), where the proof instead is presented as a sequence of tactics.

Both the theorem and the program (proof) is interactively synthesised by the user. At each stage the user can inspect the type of goals (place-holders in the proof), and their possible completions. During this process the user benefits from ALF's window interface. Several windows are maintained including a "scratch area", where the current incomplete proof is displayed, and a "theory area", with relevant definitions and theorems from earlier developments. The proof can be built by pointing and clicking in the windows or from the menus. Seeing the proof term gives a high degree of control of the proof and may sometimes make it easier to find concise proofs. One can contrast the philosophy behind ALF, which emphasises the proofs themselves as objects of independent interest, to the more traditional view that truth is the central notion and proofs are only the tool for finding truths.

To sum up, programming (and proving!) in ALF feels much like programming in a standard functional language, but with the extra expressiveness of dependent types added. It is clear that the abstract syntax of ALF is similar to that of ordinary functional programming languages, but there are some differences in concrete syntax. For example, parentheses are used in the Pascal, rather than the ML, fashion. (This has changed in the most recent version of ALF however.)

4.4 Pattern matching in ALF

Martin-Löf's type theory is an open theory, and new constants can be introduced when there is a need. There are two ways to introduce constants: ***explicitly defined constants*** and ***implicitly defined constants*** [9], [8].

We declare an explicit constant c by giving a definition of it:

$$c = a \in A$$

For instance we can make the following explicit definitions:

$$1 = s(0) \in N$$
$$I_N = [x]x \in (N)N$$
$$plus = [x,y]natrec(x,y,[u,v]s(v))$$

Or explicit constants are just abbreviations.

We declare an implicit constant by showing what definiens it has when we apply it to its arguments. This can be done by *pattern matching* easily. For instance, after declaring $add \in (N;N)N$ as an implicit constant, we can choose Make pattern in the menu in ALF. A defining equation with a placeholder as right-hand side appears. Now we want to do case analysis on the first argument, so we select the parameter x in the left hand-side and invoke make pattern again, The equation is split in two, one for each possible constructor form of N. Now the right-hand side can be easily filled in:

$$add \in (N;N)N$$
$$add(0,y) = y$$
$$add(s(x),y) = s(add(x,y))$$

When we define a constant by pattern matching, ALF generates a list of *exhaustive* and *mutually disjoint* cases. The cases are computed using the declaration of the set former of which the type of the selected argument is an instance. This can be done under certain conditions on both the declaration of the set former and the type of the selected argument, for the explanation we refer to [8, 24].

The two ways of defining "+" presented here are just instances of two different disciplines. In the first discipline, one defines for each set once and for all an elimination rule, capturing proof by structure induction over elements of the set. These elimination rules are defined as implicit constants and are justified by reflection on the definition of the set. Having done this, all proofs involving elements of this set are defined as explicit constants. In the second discipline, only the set with its introduction rules is defined at the outset and later proofs are done by pattern matching, involving a reflection on the arguments specially adapted to the particular proposition one wants to prove. The later discipline is much easier than the first one.

When introducing a constant by pattern matching in ALF, to ensure that a recursive definition leads to a well-defined function it is necessary that there is an argument position in which the recursive call has a ***structurally smaller*** argument. The ALF version we are using does not enforce this condition. It is thus possible in ALF to make meaningless recursive definitions, such as $f(x) = f(x)$. The user must check that recursive definitions are well-formed.

4.5 Some experience in the proof checking in this thesis

Writing ALF proofs on the machine is very much like we write proofs on paper by hand. But the version the author used in this thesis does not have garbage collection and consumes a huge memory. Running ALF can slowdown the system on which ALF is running. Therefore the author had a hard experience in the proof checking in this thesis.

All the proofs in this thesis were done on a powerful public machine in the department. To avoid crash with other people on the machine, the author had to avoid running ALF during working time. Sometimes ALF crashed when the scratch area was too big and the proofs in the scratch area could not be reloaded. Therefore the author had to restart those proofs in the scratch area. Some lemmas took ALF hours to check, and they are too big to ALF. The author apologies to the people in the department for the inconvenience the author brought when running ALF.

Chapter 5

Formalise Abstract Reduction Systems in ALF

In this chapter we shall formalise those basic notions in abstract reduction systems which will be used in sequel. All these notions are given in chapter 2.

5.1 Implementing confluence of ARS

An abstract reduction system is a pair (A, R), where $A : set$ and $R : (A; A)Set$ is one step reduction rule. We need to define the transitive closure R^+ and reflexive and transitive closure R^*.

The transitive closure R^+ is defined informally as:

If $R(a, b)$ then $R^+(a, b)$, and if $R^+(a, b)$ and $R^+(b, c)$ then $R^+(a, c)$. The definition of R^+ in ALF is very direct:

$$
\begin{aligned}
&\mathit{StepPlus} : (A : \mathit{Set}; R : (A; A)\mathit{Set}; A; A)\mathit{Set} \\
&\quad \mathit{OneStep}(A : \mathit{Set}; R : (A; A)\mathit{Set}; a, b : A; R(a, b))\mathit{StepPlus}(A, R, a, b) \\
&\quad \mathit{ManySteps}(A : \mathit{Set}; R : (A; A)\mathit{Set}; a, b, c : A; \mathit{StepPlus}(A, R, a, b); \\
&\qquad\qquad\qquad \mathit{StepPlus}(A, R, b, c))\mathit{StepPlus}(A, R, a, c)
\end{aligned}
$$

where we suppose $(A : Set; R : (A; A)Set)$ is an ARS, $StepPlus(A, R, a, b)$ denotes the transitive closure $R^+(a, b)$. It has two constructors, *OneStep* and *ManySteps*, which corresponds to the two conditions in the informal definition.

Because we are using a monomorphic version of ALF, there are a lot of type information in the ALF codes. But we can hide some information which are not important when displaying the definitions. For instance, we can hide some arguments which are not interesting or their types can be derived from the expressions, e.g. A in $StepPlus(A, R, a, b)$, and A, R, a, b in the constructor $OneStep$ in the above definition:

$$\begin{aligned}&StepPlus : (R : (A; A)Set; A; A)Set\\&\quad OneStep(R(a, b))StepPlus(R, a, b)\\&\quad ManySteps(StepPlus(R, a, b); StepPlus(R, b, c))StepPlus(R, a, c)\end{aligned}$$

We can further hide the argument R in $StepPlus(R, a, b)$ and write it just $StepPlus(a, b)$ if R is not important. To improve readability we shall hide some arguments when writing ALF codes.

The reflexive and transitive closure of R is defined as:

For any $a \in A$, $R^*(a, a)$ and if $R^+(a, b)$ then $R^*(a, b)$.

We have the very direct translation of R^* in ALF:

$$\begin{aligned}&StepStar : (R : (A; A)Set; A; A)Set\\&\quad ZeroStep : StepStar(R, a, a)\\&\quad SeverlSteps : (StepPlus(R, a, b))StepStar(R, a, b)\end{aligned}$$

We have hidden some arguments in the above definition.

Having formalised R^+ and R^*, we can now formalise the notion of confluence.

$h \in A$ is Weakly confluent can be defined as an explicit constant in ALF:

$$\begin{aligned}WeakConfAt \equiv [h]Forall(A, [h_1]Forall(A, [h2]Imply(And(R(h, h_1), R(h, h_2)),\\ Exists(A, [h_3]And(StepStar(h_1, h_3), StepStar(h_2, h_3)))))) : (h : A)Set\end{aligned}$$

Then the reduction relation R is weakly confluent is written as an explicit constant:

$$WeakConf \equiv Forall(A, [h]WeakConfAt(h)) : Set$$

We are writing the ALF codes. If $WeakConfAt$ is written in a friendly way, it would be:

$$\begin{aligned}WeakConfAt(h) \equiv \forall h_1, h_2 \in A((R(h, h_1) \wedge R(h, h_2)) \to\\ \exists h_3 \in A(StepStar(h_1, h_3) \wedge StepStar(h_2, h_3)))\end{aligned}$$

Similarly we define $h \in A$ is confluent as an explicit constant:

$$ConfluentAt(h) \equiv \forall h_1, h_2 \in A((StepStar(h, h_1) \wedge StepStar(h, h_2)) \to \\ \exists h_3 \in A(StepStar(h_1, h_3) \wedge StepStar(h_2, h_3)))$$

The reduction relation R is Confluent is defined as an explicit:

$$Confluence \equiv \forall h \in AConfluentAt(h)$$

5.2 Implementing termination of ARS

The notion of *strongly normalisation* is defined as a family of sets inductively in ALF by the following rules:

Formation rule:

$$\frac{a : A}{SN(a) : Set}$$

The formation rule says there is a set SN for any $a \in A$.

Introduction rule:

$$\frac{a : A}{SNintr : (a : A; h : (b : A; R(a, b))SN(b))SN(a)}$$

This is a typical constructive way to describe infinite objects, which says an element a is strongly normalising if whenever it is one step reduced to a term b, b is also strongly normalising. This is a recursive definition. $SNintr$ is the constructor. We will use both $SN(a)$ and $a \in SN$ to express that term a is strongly normalising.

We have hidden the arguments A and R in the introduction rule.

Elimination rule:

$$\frac{\begin{array}{l} C : (t : A; SN(t))Set \\ d : (x : A; b : (y : A; R(x, y))SN(y); b_1 : (y : A; a : R(x, y))C(y, b(y, a))C(x, SNintr(x, b)) \\ t : A \\ sn : SN(t) \end{array}}{SNelim(C, b, t, sn) : C(t, sn)}$$

Suppose that C is a proposition on $SN(a)$ for any term $a \in A$. The elimination rule says that to prove that C is true for any element $sn \in SN(t)$ for any term $t \in A$, we need to prove for any strongly normalising term x, whenever x is one step reduced to y, and there is a proof of $C(y, b(y, a))$, we can get a proof of $C(x, SNintr(x, b))$, where b_1 is the induction hypothesis.

We will use the non-dependent version of the recursor:

$$\frac{\begin{array}{l} P : (a : A)Set \\ h : (m : A; (n : A; R(m,n))SN; (n : A; R(m,n))P(n))P(m) \\ m_1 : A \\ sn : SN(m_1) \end{array}}{RecSN(P, h, m_1, sn) : P(m_1)}$$

to simulate induction over the length of the longest reduction of a strongly normalising term.

Thinking of $RecSN$ as an induction principle, it says that SN is the smallest set of terms which are closed under one step reduction.

In later chapters we need to prove propositions like $SN(a)$ implies $SN(a')$. To prove such propositions using the induction principle above we can try to find a predicate P such that $SN(a)$ implies P(a), and P(a) implies $SN(a')$. To prove $SN(a)$ implies P(a), by the induction principle we need only to prove P is closed under one step reduction. We will use this technique to prove some lemmas in chapter 9.

Alternatively, we define $a \prec b$ if $b \to a$. Then a reduction $\to$ is strongly normalising if and only if the order $\prec$ is well founded. Therefore the induction principle $RecSN$ is just the well founded induction principle [31].

Chapter 6

Formalising $\lambda\sigma$ in ALF

In this chapter we give the implementations of $\lambda\sigma$ and $\lambda\sigma_0$, an invariant of $\lambda\sigma$.

6.1 Implementing $\lambda\sigma$

σ is a two-sorts substitution, so we need to formalise terms and substitutions separately. Both of them are inductively defined sets, and they are simultaneously defined. We shall give the ALF presentation after its informal definition for every set in section 3.1.

The set of $\lambda\sigma$-terms is defined as:

$$\Lambda\sigma^t ::= 1 \mid \Lambda\sigma^t\Lambda\sigma^t \mid \lambda\Lambda\sigma^t \mid \Lambda\sigma^t[\Lambda\sigma^s]$$

This set is inductively defined in ALF by the following introduction rule:

$$\begin{aligned}
&SgTms : Set\\
&\quad SgV1 : SgTms\\
&\quad SgApp : (a, b : SgTms)SgTms\\
&\quad SgLam : (a : SgTms)SgTms\\
&\quad SgProp : (a : SgTms; s : SgSubs)SgTms
\end{aligned}$$

SgTms is the the name of the set $\Lambda\sigma^t$. There are four constructors in the introduction rule, which correspond to four ways building σ-terms. *SgSubs* in the fourth constructor is the set of substitutions in $\lambda\sigma$, which is defined simultaneously.

The set of substitutions in $\lambda\sigma$ is defined as:

$$\Lambda\sigma^s ::= id \,|\!\uparrow\!|\, \Lambda\sigma^t \cdot \Lambda\sigma^s \mid \Lambda\sigma^s \circ \Lambda\sigma^s$$

This is defined in ALF by the following introduction rule:

$$\begin{aligned}
&SgSubs : Set\\
&\quad SgId : SgSubs\\
&\quad SgShift : SgSubs\\
&\quad SgAppe : (a : SgTms; s : SgSubs)SgSubs\\
&\quad SgCom : (s, t : SgSubs)SgSubs
\end{aligned}$$

To improve readability we will feel free to use $\Lambda\sigma^s$ and $\Lambda\sigma^t$ for SgTms and SgSubs respectively.

$\Lambda\sigma^t$ and $\Lambda\sigma^s$ are mutually defined. So their elimination rules are defined mutually too. This means that we need to use simultaneous induction when proving a property for one set.

We write the elimination rules of $SgTms(\Lambda\sigma^t)$ and $SgSubs(\Lambda\sigma^s)$ here in ALF code style.

Elimination rule for the set $SgTms$:

$$\begin{aligned}
&SgTms\_elimi : (C_1 : (SgTms)Set; \quad C_2 : (SgSubs)Set\\
&\quad e_1 : C_1(SgV1)\\
&\quad e_2 : (a : SgTms; b : SgTms; C_1(a); C_1(b))C_1(SgApp(a, b))\\
&\quad e_3 : (a : SgTms; C_1(a))C_1(SgLam(a))\\
&\quad e_4 : (a : SgTms; s : SgSubs; C_1(a); C_2(s))C_1(SgProp(a, s))\\
&\quad d_1 : C_2(SgId)\\
&\quad d_2 : C_2(SgShift)\\
&\quad d_3 : (a : SgTms; s : SgSubs; C_1(a); C_2(s))C_2(SgAppe(a, s))\\
&\quad d_4 : (s : SgSubs; t : SgSubs; C_2(s); C_2(t))C_2(SgCom(s, t))\\
&\quad x : SgTms)C_1(x)
\end{aligned}$$

Elimination rule for the set *SgSubs*:

$$
\begin{aligned}
& SgSubs\_elimi : (C_1 : (SgTms)Set; \quad C_2 : (SgSubs)Set \\
& \quad e_1 : C_1(SgV1) \\
& \quad e_2 : (a : SgTms; b : SgTms; C_1(a); C_1(b))C_1(SgApp(a,b)) \\
& \quad e_3 : (a : SgTms; C_1(a))C_1(SgLam(a)) \\
& \quad e_4 : (a : SgTms; s : SgSubs; C_1(a); C_2(s))C_1(SgProp(a,s)) \\
& \quad d_1 : C_2(SgId) \\
& \quad d_2 : C_2(SgShift) \\
& \quad d_3 : (a : SgTms; s : SgSubs; C_1(a); C_2(s))C_2(SgAppe(a,s)) \\
& \quad d_4 : (s : SgSubs; t : SgSubs; C_2(s); C_2(t))C_2(SgCom(s,t)) \\
& \quad x : SgSubs)C_2(x)
\end{aligned}
$$

When we reduce a term in σ, actually we are using two kinds of reductions, one is for terms and the other is for substitutions, although one might not realize it in an informal reasoning. In fact, we have two kinds of reduction rules in definition 3.1.1, the left column are the rules how terms (of type SgTms) are reduced and the right column are the rules how the substitutions (of type SgSubs) are reduced, which need to be formalised separately.

To improve readability we will use the notations in chapter 3 instead of those in ALF codes. For instance we shall use $s \circ t$ instead of $SgCom(s,t)$.

The set of term reduction rules is defined in ALF as follows:

$$
\begin{aligned}
& SgTmsRules : (\Lambda\sigma^t; \Lambda\sigma^t)Set \\
& \quad VrId : SgTmsRules(1[id], 1) \\
& \quad VrCons : SgTmsRules(1[a \cdot s], a) \\
& \quad App : SgTmsRules((a \cdot b)[s], (a[s])(b[s])) \\
& \quad Abs : SgTmsRules((\lambda a)[s], \lambda(a[1 \cdot (s \circ \uparrow)])) \\
& \quad Clos : SgTmsRules((a[s])[t], a[s \circ t])
\end{aligned}
$$

This set has 5 constructors corresponding to 5 term reduction rules. We have hidden some arguments in the constructors above. For instance, the constructor corresponding to the rule

$$VrCons : \quad 1[a \cdot s] \to a$$

would have been written as

$$VrCons : (a : \Lambda\sigma^t; s : \Lambda\sigma^s) SgTmsRules(1[a \cdot s], a)$$

We shall hide some uninteresting arguments later on to make expressions in ALF more elegant.

The set of substitution reduction rules is defined in ALF as follows:

$$\begin{aligned} &SgSubsRules : (\Lambda\sigma^s; \Lambda\sigma^s) Set \\ &\quad IdL : SgSubsRules(id \circ s, s) \\ &\quad ShiftId : SgSubsRules(\uparrow \circ id, \uparrow) \\ &\quad ShiftCons : SgSubsRules(\uparrow \circ (a \cdot s), s) \\ &\quad Map : SgSubsRules((a \cdot s) \circ t, a[t] \cdot (s \circ t)) \\ &\quad Ass : SgSubsRules((s_1 \circ s_2) \circ s_3, s_1 \circ (s_2 \circ s_3)) \end{aligned}$$

This set has 5 constructors corresponding to five substitution reduction rules.

Then we have one step reductions for terms and substitutions separately.

One step reduction for substitutions is defined as a set inductively:

$$\begin{aligned} &SgSubsOneStep : (\Lambda\sigma^s; \Lambda\sigma^s) Set \\ &\quad UsingSubsRules : (SgSubsRules(s, t)) SgSubsOneStep(s, t) \\ &\quad SgAppeComptL : (SgTmsOneStep(a, b)) SgSubsOneStep(a \cdot s, b \cdot s) \\ &\quad SgAppeComptR : (SgSubsOneStep(s, t)) SgSubsOneStep(a \cdot s, a \cdot t) \\ &\quad SgComComptL : (SgSubsOneStep(s_1, s_2)) SgSubsOneStep(s_1 \circ t, s_2 \circ t) \\ &\quad SgComComptR : (SgSubsOneStep(s_1, s_2)) SgSubsOneStep(t \circ s_1, t \circ s_2) \end{aligned}$$

We have hidden some arguments in here again. For instance, we would have written the second constructor as:

$$SgAppeComptR : (a : \Lambda\sigma^t; s, t : \Lambda\sigma^s; SgSubsOneStep(s, t)) SgSubsOneStep(a \cdot s, a \cdot t)$$

if all the arguments are displayed.

The first rule says if $s \rightarrow t$ is a rule, then s one step reduce to t. The second and third constructors define the compatible reduction rules for the operation "$\cdot$". The second constructor says if a one step reduce to b, then $a \cdot s$ one step reduce to $b \cdot s$, where a one step reduce to b is defined below(a, b are terms). This means we must define two kinds of one step reduction simultaneously.

$SgTmsOneStep : (\Lambda\sigma^t; \Lambda\sigma^t)Set$

$UsingTmsRules : (SgTmsRules(a, b))SgTmsOneStep(a, b)$

$SgAppComptL : (SgTmsOneStep(a_1, a_2))SgTmsOneStep(a_1 b, a_2 b)$

$SgAppComptR : (SgTmsOneStep(b_1, b_2))SgTmsOneStep(ab_1, ab_2)$

$SgLamCompt : (SgTmsOneStep(a, b))SgTmsOneStep(\lambda a, \lambda b)$

$SgPropComptL : (SgTmsOneStep(a, b))SgTmsOneStep(a[s], b[s])$

$SgPropComptR : (SgSubsOneStep(s, t))SgTmsOneStep(a[s], a[t])$

After defining the set of terms $SgTms$ and one step reduction relation $SgTmsOneStep$ on $SgTms$, we get the relation $\rightarrow^+_{\Lambda\sigma^t}$ $(StepPlus(SgTms, SgTmsOneStep, SgTms, SGTms))$ and $\rightarrow^*_{\Lambda\sigma^t}$ $(StepStar(SgTms, SgTmsOneStep, SgTms, SGTms))$.

We will use $\rightarrow^+_\sigma$ for both the terms reductions and substitutions reductions.

6.2 Implementing $\lambda\sigma_0$

The calculus $\lambda\sigma_0$ is an one-sort calculus. The set $\Lambda\sigma_0$ is defined as:

$SgzTms : Set$

$v : SgzTms$

$id : SgzTms$

$shift : SgzTms$

$lam : (SgzTms)SgzTms$

$appe : (SgzTms; SgzTms)SgzTms$

$com : (SgzTms; SgzTms)SgzTms$

We will feel free to use more elegant notations given in chapter 3, e.g. $a \cdot b$ instead of $appe(a, b)$, $a \circ b$ instead of $com(a, b)$.

Then we have the elimination rule of the set *SgzTms*:

$$
\begin{aligned}
&\mathit{SgzTms\_elimi} : (C : (\mathit{SgzTms})\mathit{Set} \\
&\quad e_1 : C(1) \\
&\quad e_2 : C(id) \\
&\quad e_3 : C(\uparrow) \\
&\quad e_4 : (a : \mathit{SgzTms})C(\lambda(a)) \\
&\quad e_5 : (a : \mathit{SgzTms}; b : \mathit{SgzTms}; C(a); C(b))C(a \cdot b) \\
&\quad e_6 : (a : \mathit{SgzTms}; b : \mathit{SgzTms}; C(a); C(b))C(a \circ b)) \\
&\quad x : \mathit{SgzTms})C(x)
\end{aligned}
$$

The reduction rules is an inductively defined set with 8 constructors corresponding to the eight rules in definition 3.1.2:

$$\mathit{SgzRules} : (\mathit{SgzTms}; \mathit{SgzTms})\mathit{Set}$$

We have omitted the constructors.

One step reduction in σ_0 is defined in the same way as *SgSubsOneStep*.

$$\mathit{SgzOneStep} : (\mathit{SgzTms}; \mathit{SGzTms})\mathit{Set}$$

Then we have transitive closure and reflexive and transitive closure of the reduction *SgzOneStep*.

Chapter 7

Formalising A Context Calculus for σ_0 in ALF

In this chapter we shall formalise in ALF all the notions informally given in [11].

Many notions were taken for granted and not introduced, and many lemmas were also taken for granted from the intuition and left unproved in [11]. To formalise the proofs in [11] we had to rewrite all the intuitions and informal notions, and to check a lot of details. Sometimes we had to change the implementation to make the proofs go through. We shall discuss some of the implementations during the process of the formalisation in this chapter.

7.1 Definition of the context

The basic idea of the context calculus is to think of a term t as a "context" with multi-holes filled by its sub-terms, and to see the machinery of these contexts while reducing t.

Many notions about the context for σ_0 have been informally presented in [11]. We will give all the formal definitions in this section.

Definition 7.1.1 *Contexts with multi-holes are given inductively by:*

$C ::= \Box_n \mid 1 \mid id \mid \uparrow \mid \lambda C \mid C \cdot D \mid C \circ D$

where $n \geq 1$ *and* $\Box_n$ *denotes a hole.*

Contexts are defined as a set with seven constructors in ALF:

$$\begin{aligned}
&\mathcal{C} \in Set\\
&\quad hole \in (n \in N)\mathcal{C}\\
&\quad v_c \in \mathcal{C}\\
&\quad id_c \in \mathcal{C}\\
&\quad shift_c \in \mathcal{C}\\
&\quad lam_c \in (C : \mathcal{C})\mathcal{C}\\
&\quad appe_c \in (C, D : \mathcal{C})\mathcal{C}\\
&\quad com_c \in (C, D : \mathcal{C})\mathcal{C}
\end{aligned}$$

where $v_c, id_c, shift_c, lam_c, appe_c$ and com_c are constructors (notations in ALF) corresponding to $1, id, \uparrow, \lambda, \cdot$ and $\circ$ respectively. We shall use v_c, id and $\uparrow$ to denote $1, id$ and $\uparrow$ respectively if no confusion arises. $com_c(C, D)$ will be denoted as $C \circ_c D$ or $C \circ D$, $appe_c(C, D)$ as $C \cdot D$ and $lam_c(C)$ as $\lambda_c(C)$ or $\lambda(C)$ when it is clear from the context. We shall use $\Box_n$ to denote the constructor $hole(n)$.

We need to specify some sub-contexts in a context. This is done by the position of the sub-context, which is a list of the set $\{1, 2\}$.

$$\begin{aligned}
&SubCont : (C \in \mathcal{C}; \gamma \in \mathcal{L})\mathcal{C}\\
&\quad SubCont(C, nil) = C\\
&\quad SubCont(\Box_m, \{1, 2\}\gamma) = v_c\\
&\quad SubCont(id_c, \{1, 2\}\gamma) = id_c\\
&\quad SubCont(shift_c, \{1, 2\}\gamma) = shift_c\\
&\quad Subcont(lam_c(C), \{1, 2\}\gamma) = Subcont(C, \gamma)\\
&\quad SubCont(appe_c(C, D), 1\gamma) = SubCont(C, \gamma)\\
&\quad SubCont(appe_c(C, D), 2\gamma) = SubCont(D, \gamma)\\
&\quad SubCont(com_c(C, D), 1\gamma) = SubCont(C, \gamma)\\
&\quad SubCont(com_c(C, D), 2\gamma) = SubCont(D, \gamma)
\end{aligned}$$

where $\{1, 2\}$ means 1 or 2.

A list on $\{1, 2\}$ is also called an occurrence. The set of occurrences will be denoted as $\mathcal{L}$.

We will use the notation C/γ for $SubCont(C, \gamma)$, i.e. the sub-context of C at γ. $\gamma' \prec \gamma$ will denote γ' is a proper prefix of γ.

$SubCont(C,\gamma)$ is a partial function on γ, but we have to define $SubCont$ as a total function on occurrences γ in ALF. We need a number of partial functions on the set of occurrence $\mathcal{L}$, we will define them as total functions as well in ALF.

Remark 7.1.2 *We could deal with partial functions by defining "Option" in the following way:*

$$\begin{aligned}
&Option : (A : Set)Set\\
&\quad some : (A : set; a : A)Option(A)\\
&\quad none : (A : Set)Option(A)
\end{aligned}$$

Then a partial function $f : A \rightarrow B$ can be defined as a total function $f' : A \rightarrow Option(B)$.

It is easy to see that:

For any $x \in A$ either $f(x) = none(B)$ or $\exists b \in B(f(x) = some(b))$.

Similarly we can define the sub-terms of a term by its occurrences of the sub-terms.

$$\begin{aligned}
&SubTm : (s \in \Lambda\sigma_0; \gamma \in \mathcal{L})\Lambda\sigma_0\\
&\quad SubTm(t, nil) = t\\
&\quad SubTm(1, \{1,2\}\gamma) = 1\\
&\quad SubTm(id, \{1,2\}\gamma) = id\\
&\quad SubTm(\uparrow, \{1,2\}\gamma) = \uparrow\\
&\quad SubTm(\lambda(s), \{1,2\}\gamma) = SubTm(s,\gamma)\\
&\quad SubTm(s \cdot t, 1\gamma) = SubTm(s,\gamma)\\
&\quad SubTm(s \cdot t, 2\gamma) = SubTm(t,\gamma)\\
&\quad SubTm(s \circ t, 1\gamma) = SubTm(s,\gamma)\\
&\quad SubTm(s \circ t, 2\gamma) = SubTm(t,\gamma)
\end{aligned}$$

$SubTm(s,\gamma)$ will be denoted as s/γ.

7.2 Substitutions of the context

There are several ways to think of a term t as a context filled with its sub-terms. We first define the substitutions of contexts with tuples of terms:

$$
\begin{aligned}
&SubstCont \in (C \in \mathcal{C}; u \in \Lambda\sigma_0^n)\Lambda\sigma_0\\
&\quad SubstCont(\Box_m, u) = Proj(n, m, u)\\
&\quad SubstCont(id_c, u) = id\\
&\quad SubstCont(v_c, u) = 1\\
&\quad SubstCont(shift_c, u) = \uparrow\\
&\quad SubstCont(lam_c(C), u) = \lambda(SubCont(C, u))\\
&\quad SubstCont(appe_c(C, D), u) = SubstCont(C, u) \cdot SubstCont(D, u)\\
&\quad SubstCont(com_c(C, D), u) = SubCont(C, u) \circ SubstCont(D, u)
\end{aligned}
$$

where

$$
\begin{aligned}
&Proj : (n, k \in N; u \in \Lambda\sigma_0^n)\Lambda\sigma_0\\
&\quad Proj(0, i, u) = 1\\
&\quad Proj(n_1 + 1, 0, < a, b >) = b\\
&\quad Proj(n_1 + 1, n + 1, < a, b >) = Proj(n_1, n, a)
\end{aligned}
$$

We shall also use u_k to denote $Proj(n, k, u)$, $C[u]$ for $Subst(C, u)$.

It is natural to require that $n \geq N_c(C)$ in the definition of *SubstCont*, where $N_c(C) = max\{m : \Box_m \in C\}$. But we will have problems when proving intensional equality properties of *SubstCont* and matching the proof object $n \geq N_c$ if this argument is added in the definition of *SubstCont*. We have decided to give up the conditions by extending the definition. The argument n is omitted in the definition of *SubstCont*.

Notation 7.2.1 *Let C be a context and $q \geq 0$. C_q denotes the context obtained from C by renaming the holes $\Box_k$ as $\Box_{k+q}$.*

$$LiftCont : (C \in \mathcal{C}; q \in N)\mathcal{C}$$

Let $u = (u_1, \cdots, u_m)$ and $v = (v_1, \cdots, v_n)$. The juxtaposition of u and v will be denoted by $u@v = (u_1, \cdots, u_m, v_1, \cdots, v_n)$. we will denote the length of u by $Lg(u)$ or $|u|$.

The following lemma states some basic facts about the substitution $C[u]$:

Lemma 7.2.2 *Suppose $u = (u_1, \cdots, u_m)$ and $v = (v_1, \cdots, v_n)$*

1. *$Proj(m+n, m+k, u@v) = Proj(n, k, v)$.*

2. *$C[u@v] = C[u]$ if $N_c(C) \leq Lg(u)$.*

3. *$C_m[u@v] = C[v]$.*

4. *If $K_c(C) = 0$, then $C[u] = C[v]$ for any u, v.*

where $K_c(C)$ is the number of the holes in C.

Intuitively they are all true, but we need to check a lot of cases by induction to be able to formally prove they are true. For all the ALF proofs the reader is referred to http://www.dcs.gla.ac.uk/people/personal/qiao/papers/sig0sn.ps.

For any terms a, b, we will have $Proj(m+1, m, u@a) = a$ and $Proj(m+2, m, u@ < a, b >) = a$.

In the process of proof checking we need to make many trivial lemmas explicit:

Lemma 7.2.3 *Suppose that s, s_1, s_2, t, t_1, t_2 are terms.*

1. *$\lambda(s) = \lambda(t)$ iff $s = t$.*

2. *$s_1 \cdot t_1 = s_2 \cdot t_2$ iff $s_1 = s_2$ and $t_1 = t_2$.*

3. *$s_1 \circ t_1 = s_2 \circ t_2$ iff $s_1 = s_2$ and $t_1 = t_2$.*

Substitutions in contexts is a basic operation in the context calculus. We need also the following notations: $C\{\gamma \leftarrow D\}$, the context obtained by replacing in C the sub-context C/γ by the context D, and $s\{\gamma \leftarrow t\}$, the term obtained by replacing in s the sub-term s/γ by the term t.

$$
\begin{aligned}
&\mathit{SubstSubcont} : (C \in \mathcal{C}; \gamma \in \mathcal{L}; B \in \mathcal{C})\mathcal{C}\\
&\quad \mathit{SubstSubcont}(C, \mathit{nil}, B) = B\\
&\quad \mathit{SubstSubcont}(\Box_m, \{1,2\}\gamma, B) = \Box_m\\
&\quad \mathit{SubstSubcont}(\mathit{id}_c, \{1,2\}\gamma, B) = \mathit{id}_c\\
&\quad \mathit{SubstSubcont}(v_c, \{1,2\}\gamma, B) = v_c\\
&\quad \mathit{SubstSubcont}(\mathit{shift}_c, \{1,2\}\gamma, B) = \mathit{shift}_c\\
&\quad \mathit{SubstSubcont}(\mathit{lam}_c(C), \{1,2\}\gamma, B) = \mathit{lam}_c(\mathit{SubstSubcont}(C, \gamma, B))\\
&\quad \mathit{SubstSubcont}(\mathit{appe}_c(C, D), 1\gamma, B) = \mathit{appe}_c(\mathit{SubstSubcont}(C, \gamma, B), D)\\
&\quad \mathit{SubstSubcont}(\mathit{appe}_c(C, D), 2\gamma, B) = \mathit{appe}_c(C, \mathit{SubstSubcont}(D, \gamma, B))\\
&\quad \mathit{SubstSubcont}(\mathit{com}_c(C, D), 1\gamma, B) = \mathit{com}_c(\mathit{SubstSubcont}(C, \gamma, B), D)\\
&\quad \mathit{SubstSubcont}(\mathit{com}_c(C, D), 2\gamma, B) = \mathit{com}_c(C, \mathit{SubstSubcont}(D, \gamma, B))
\end{aligned}
$$

$\mathit{TmtoCont}(s)$ will denote the context when thinking of term s as a context without any hole. We shall use $C\{\gamma \leftarrow t\}$ instead of $C\{\gamma \leftarrow \mathit{TmtoCont}(t)\}$.

$s\{\gamma \leftarrow t\}$ is defined in the same way:

$$
\begin{aligned}
&\mathit{SubstSubtm} : (s \in \Lambda\sigma_0; \gamma \in \mathcal{L}; t \in \Lambda\sigma_0)\Lambda\sigma_0\\
&\quad \mathit{SubstSubtm}(s, \mathit{nil}, t) = t\\
&\quad \mathit{SubstSubtm}(1, \{1,2\}\gamma, t) = 1\\
&\quad \mathit{SubstSubtm}(\mathit{id}, \{1,2\}\gamma, t) = \mathit{id}\\
&\quad \mathit{SubstSubtm}(\uparrow, \{1,2\}\gamma, t) = \uparrow\\
&\quad \mathit{SubstSubtm}(\lambda(s), \{1,2\}\gamma, t) = \lambda(\mathit{SubstSubtm}(C, \gamma, t)\\
&\quad \mathit{SubstSubtm}(s_1 \cdot s_2, 1\gamma, t) = \mathit{SubstSubtm}(s_1, \gamma, t) \cdot s_2\\
&\quad \mathit{SubstSubtm}(s_1 \cdot s_2, 2\gamma, t) = s_1 \cdot \mathit{SubstSubtm}(s_2, \gamma, t)\\
&\quad \mathit{SubstSubtm}(s_1 \circ s_2, 1\gamma, t) = \mathit{SubstSubtm}(s_1, \gamma, t) \circ s_2\\
&\quad \mathit{SubstSubtm}(s_1 \circ s_2, 2\gamma, t) = s_1 \circ \mathit{SubstSubtm}(s_2, \gamma, t)
\end{aligned}
$$

We are considering both substitutions on variables (holes) and on "positions", where we only substitute some occurrences of a variable. We shall have several versions of substitution lemmas.

In σ_0 a term is called a W-term if no λ occurs in it. Otherwise it is called an L-term. It is obvious that a term is either a W-term or an L-term. Many lemmas of the context calculus were proved by analysing if a term is a W-term or L-term. We have the following lemma about W-terms and L-terms:

Lemma 7.2.4 *Suppose that s, t are terms, γ, δ are occurrences.*

- *Any term s is either a W-term or an L-term.*
- *If s is a W-term then for any occurrence γ, s/γ is a W-term.*
- *If s/γ is an L-term for some occurrence γ , then s is L-term.*
- *If t is an L-term and s/γ is an L-term, then $s\{\gamma \leftarrow t\}$ is an L-term.*
- *If $s\{\gamma \leftarrow t\}$ is an L-term for some W-term t, then s is an L-term.*
- *Suppose s/γ is a W-term and t is a W-term. If $s\{\gamma \leftarrow t\}/\delta$ is an L-term, then s/δ is also an L-term.*

Now we define the relation between contexts and terms:

Definition 7.2.5 *A context C* relative to *s is a context such that $s = C[u]$ for some u.*

The constructive definition of this relation is defined inductively on the structure of C as follows:

$$
\begin{aligned}
&ContForTm : (C : \mathcal{C}; s : \Lambda\sigma_0)Set\\
&\quad ContForTm(\Box_m, s) = I(s, s)\\
&\quad ContForTm(id, s) = I(s, id)\\
&\quad ContForTm(v_c, s) = I(s, 1)\\
&\quad ContForTm(\uparrow, s) = I(s, \uparrow)\\
&\quad ContForTm(\lambda_c(C), s) = \exists h(I(s, \lambda(h)) \wedge ContForTm(C, h))\\
&\quad ContForTm(C \cdot D, s) = \exists a, b(I(s, a \cdot b) \wedge ContForTm(C, a) \wedge ContForTm(D, b))\\
&\quad ContForTm(C \circ_c D, s) = \exists a, b(I(s, a \circ b) \wedge ContForTm(C, a) \wedge ContForTm(D, b))
\end{aligned}
$$

where I is the intensional equality.

A hole $\Box_m$ in a context C relative to s is called a W-hole if the corresponding sub-term u_m is a W-term, otherwise it is called an L-hole.

The following lemma is used to check the properties of what will be called very good inflation.

Lemma 7.2.6 *Suppose that C is a context, s, t are terms, γ, δ are occurrences.*

1. *C/γ is a hole if and only if C_q/γ is a hole.*

2. *If $C\{\gamma \leftarrow t\}/\delta$ is a W-hole (L-hole), then C/δ is a W-hole (L-hole).*

3. *If C/γ is a hole, C is a context for $C[u]$, then there exists $k \leq |u| = n$ such that $C[u]/\gamma = Proj(n, k, u)$.*

4. *If C/γ is a hole, then there exists $k \leq n$ such that for all u, $C[u]/\gamma = u_k$.*

Remark 7.2.7 *In the above lemma, the witness k in 3 does not depend on u, so if C/γ is a hole, for any u, we have a uniform k such that $C[u]/\gamma = u_k$, and $C[v]/\gamma = v_k$. This is needed when we have $C[u\tilde{o}w]/\gamma = proj(n, k, u\tilde{o}w)$, we want the facts that $C[u]/\gamma = u_k$ and $C[w]/\gamma = w_k$.*

The following lemma states the relation between sub-contexts and the corresponding sub-terms.

Lemma 7.2.8 *Suppose that C, D are contexts, s, t are terms, u is any tuple of terms and γ is an occurrence.*

- *C is a context relative to s iff $\lambda_c(C)$ is a context to $\lambda(s)$.*
- *$C \cdot D$ is a context relative to $s \cdot t$ iff C is to s and D is to t.*
- *$C \circ_c D$ is a context relative to $s \circ t$ iff C is a context relative to s and D is to t.*
- *If C is a context relative to s with u, then C_q is a context relative to s with $v@u$ for any $v = (v_1, \cdots, v_q)$.*
- *If $D[u] = E[u]$, then $C\{\gamma \leftarrow D\}[u] = C\{\gamma \leftarrow E\}[u]$.*
- *If C is a context relative to s then $C\{\gamma \leftarrow t\}$ is a context relative to $s\{\gamma \leftarrow t\}$.*

Definition 7.2.9 *An inflation of s is a pair (C, w) where C is a context and w an n-tuple of W-term such that there is a n-tuple of terms u which satisfies $s = C[u]$.*

This is defined as a relation in ALF as follows:

$Inflation(s, C, u, w)$ iff $I(s, C[u]) \wedge (n \geq N_c(C)) \wedge WtmTuple(w)$ where $WtmTupl(w)$ means w is a W-Tuple, i.e. a tuple of W-terms. We have hidden the argument n, i.e. $Lg(u)$.

We shall also say (s, C, u, w) or (C, u, w) is an inflation.

The result of the inflation (s, C, u, w) is $s' = C[u']$ where u' is given by:

- $u'_k = w_k$ if u_k is a W-term.
- $u'_k = u_k \circ w_k$ otherwise.

We shall also call the result $s' = C[u']$ as an increment of s.

We define a function in ALF to express this prime operation:

$PriTuple : (u, w : \Lambda\sigma_0^n)\Lambda\sigma_0^n$

$PriTuple(0, u, w) = one$

$PriTuple(n_1 + 1, < a, b >, < a_1, b_1 >) =< PriTuple(n_1, a, a_1), IfThEl(b, b_1, b \circ b_1)) >$

where

$IfThEl : (s \in \Lambda\sigma_0; t_1, t_2 : \Lambda\sigma_0)\Lambda\sigma_0$

$IfThEl(s, t_1, t_2) = t_1$ if s is a W-term

$IfThEl(s, t_1, t_2) = t_2$ otherwise

We shall make the operation $'$ explicit and use $u\tilde{\circ}w$ to denote $PriTuple(u, w)$.

The following lemma states some important facts about the operation $\tilde{\circ}$:

Lemma 7.2.10 *Suppose C is a context, s, t are terms, u, u_1, u_2, w, w_1, w_2 are tuples of terms and γ is an occurrence.*

1. *If t is a W-term, then $IfThEl(s, s \circ t, t)$ is a W-term.*
2. *$IfThEl(s, IfThEl(s, s \circ t, s), s \circ IfThEl(s, s \circ t, t)) = s \circ t$.*
3. *$Proj(n, k, u\tilde{\circ}w)$ is a W-term (L-term) if and only if $Proj(n, k, u)$ is a W-term (L-term), where w is a W-term tuple.*
4. *$C[u\tilde{\circ}w]/\gamma$ is a W-term if and only if $C[u]/\gamma$ is a W-term, where w is a W-term tuple.*
5. *If $Proj(n, k, u\tilde{\circ}w)$ is L-term, then $Proj(n, k, u\tilde{\circ}w) = u_k \circ w_k$.*
6. *If $Proj(n, k, u\tilde{\circ}w)$ is a W-term, then $Proj(n, k, u\tilde{\circ}w) = w_k$.*
7. *If $C[u\tilde{\circ}w]/\gamma$ is L-term, then $C[u\tilde{\circ}w]/\gamma = C[u]/\gamma \circ C[w]/\gamma$.*
8. *If $C[u\tilde{\circ}w]/\gamma$ is a W-term, then $C[u\tilde{\circ}w]/\gamma = C[w]/\gamma$.*

9. *If $C[u]$ is a W-term, then $C[u\tilde{\circ}w] = C[w]$ and $C[u\tilde{\circ}w]$ is a W-term, where w is a tuple of W-terms.*

10. *If (C, w) is an inflation of $s = C[u] = C[v]$ then $C[u\tilde{\circ}w] = C[v\tilde{\circ}w]$.*

11. *If $u_1, w_1 \in \Lambda\sigma_0^m; u_2, w_2 \in \Lambda\sigma_0^n$ then $(u_1@u_2)\tilde{\circ}(w_1@w_2) = (u_1\tilde{\circ}w_1)@(u_2\tilde{\circ}w_2)$.*

We need these facts when constructing the inflations in the Preservation Theorem.

The following lemma states the relation between context operations and inflations:

Lemma 7.2.11 • *$Inflation(s, C, u, w)$ iff $Inflation(\lambda(s), \lambda_c(C), u, w)$.*

- *If $Inflation(s, C, u_1, w_1)$ and $Inflation(t, D, u_2, w_2)$, then $Inflation(s \cdot t, C \cdot D_m, u_1@u_2, w_1@w_2)$.*
- *If $Inflation(s, C, u_1, w_1)$ and $Inflation(t, D, u_2, w_2)$, then $Inflation(s \circ t, C \circ_c D_m, u_1@u_2, w_1@w_2)$.*
- *If $K_c(C) = 0$ then $Inflation(C[u], C, n, u, w)$ for any u, w, where $K_c(C)$ is the number of the holes in C.*

Now we introduce the first restriction on contexts:

Definition 7.2.12 *A context C is good for s if it is a context for s, and whenever $A \circ_c B$ is a sub-context of C and there exists a hole in A, then B must be a W-hole.*

A good context is defined as a set on C and s:

$$
\begin{aligned}
&GoodCont : (C : \mathcal{C}; s : \Lambda\sigma_0)Set\\
&\quad GoodCont(\Box_m, s) = I(s, s)\\
&\quad GoodCont(id, s) = I(s, id)\\
&\quad GoodCont(v_c, s) = I(s, 1)\\
&\quad GoodCont(\uparrow, s) = I(s, \uparrow)\\
&\quad GoodCont(\lambda_c(C), s) = \exists h(I(s, \lambda(h)) \wedge GoodCont(C, h))\\
&\quad GoodCont(C \cdot D, s) = \exists a, b(I(s, a \cdot b) \wedge GoodCont(C, a) \wedge GoodCont(D, b))\\
&\quad GoodCont(C \circ_c D, s) = \exists a, b(I(s, a \circ b) \wedge GoodCont(C, a) \wedge GoodCont(D, b) \wedge\\
&\qquad\qquad\qquad\qquad (((\exists l) IsHole(C, l)) \rightarrow \text{W-hole(D)}))
\end{aligned}
$$

where $IsHole(C, l)$ denotes C/l is a hole and W-hole(D) denotes that D is a W-hole, meaning that D is a hole and b is a W-term.

We shall also say a context C is good for a term s where we mean that C is a good context for s.

Example 7.2.13 *Let $s = (1 \cdot \uparrow) \cdot ((\uparrow \cdot id) \circ (\lambda 1))$. Then*

- *$C = (1 \cdot \Box_2) \cdot ((\uparrow \cdot \Box_4) \circ \Box_1)$ is a context relative to s, which is not good.*
- *$D = (1 \cdot \Box_2) \cdot \Box_3$ is a context relative to s, which is good.*
- *$E = \Box_5 \cdot \Box_6$ is a good context for s.*

A good Inflation will be defined as:

$GoodInflation(s, C, u, w)$ iff $GoodCont(C, s) \wedge Infaltion(s, C, u, w)$

Many facts about good contexts are needed when proving some important lemmas in section 7.3:

Lemma 7.2.14 *Suppose that C, D are contexts, and s, t, c, d, e are terms.*

- *If C is a context relative to s and $K_c(C) = 0$ then C is good for s.*
- *C is a good context for s iff $\lambda_c(C)$ is a good context for $\lambda(s)$.*
- *$C \cdot D$ is a good context for $s \cdot t$ iff C is good for s and D is good for t.*
- *$C \circ_c D$ is good for $s \circ t$ then C is good for s and D is good for t.*
- *If C is good for s, $K_c(C) = 0$, and D is good for t, then $C \circ D$ is good for $s \circ t$.*
- *If C is good for s, and t is a W-term, then $C \circ \Box_m$ is good for $s \circ t$.*
- *If $C \circ_c D$ is good for $a \circ b$, $K_c(C) \geq 1$, and E is good for e; then $E \circ_c D$ is good for $e \circ b$.*
- *If C is good for s, then C_m is good for s for any $m \in N$.*
- *If $(C \cdot D) \circ_c E$ is good for $(c \cdot d) \circ e$, then $(C \circ_c E) \cdot (D \circ_c E)$ is good for $(c \circ e) \cdot (d \circ e)$.*
- *If $(C \circ_c D) \circ_c E$ is good for $(c \circ d) \circ e$, then $D \circ E$ is good for $d \circ e$.*

- *If* $(C \circ_c D) \circ_c E$ *is good for* $(c \circ d) \circ e$ *and* $K_c(C) = 0$*, then* $C \circ_c (D \circ_c E)$ *is good for* $c \circ (d \circ e)$.

The following lemma will be needed when the main case of the Preservation theorem 7.3.8 is checked:

Lemma 7.2.15 *Suppose that* C *is a context,* s *is a term,* γ *is an occurrence and* $m \in N$.

1. *If* C *is good for* s*,* C/γ *is a W-hole, then* $C\{\gamma \leftarrow \Box_m\}$ *is good for* $s\{\gamma \leftarrow t\}$*, where* t *is a W-term; especially* $C\{\gamma \leftarrow \Box_m\}$ *is good for* s.
2. *If* C *is good for* s*,* C/γ *is a* λ*-hole, and* D *is good for* t*, then* $C\{\gamma \leftarrow D\}$ *is a good context relative to* $s\{\gamma \leftarrow t\}$.

It is proved by induction on the structure of C. The difficulty of the proof is when coming to the case $C = A \circ_c B$. We need the following facts:

Suppose that $A \circ_c B$ is good for $a \circ b$, and D is good for d, then

1. If $K_c(A) \geq 1$, then $D \circ_c B$ is good for $d \circ b$.
2. If $K_c(A) = 0$, then $A \circ_c D$ is good for $a \circ d$.
3. If B is not a W-hole, then $A \circ_c D$ is good for $a \circ d$.

Remark 7.2.16 *Quite often it is very easy to show that a lemma holds by contradiction. But we must prove the lemma by induction or case analysis on some argument, as we can't use contradiction in ALF. For instance, in the third case of the above lemma,* $A \circ_c D$ *is good because* $K_c(A) = 0$*, otherwise* $K_c(A) \geq 1$ *would implies* B *is a W-hole, a contradiction.*

We have the following facts about the context $TmtoCon(s)$, the term s seen as a context:

Lemma 7.2.17 *For any term* s *and any occurrence* γ *we have*

- $TmtoCon(s)$ *is good for* s.
- $K_c(tmtocon(s)) = 0$.

- *$Lg(tmtocon(s)) > 0$.*
- *$TmtoCon(s)/\gamma$ is not a hole for any γ.*
- *TmtoCon(s)[u]=s for any u.*

Now we introduce the second restriction on contexts:

Definition 7.2.18 *A context C for s is called very good if it is good, and whenever C/γ is a W-hole, s/δ is an L-term for any proper-prefix δ of γ.*

A very good context is defined in ALF as:
$VeryGoodCont(C, s)$ iff $GoodCont(C, s) \wedge ((\forall\gamma(IsHole(C, \gamma)) \rightarrow \forall(\delta \prec \gamma)Lterm(s/\delta)))$
In the Example 7.2.13, D is not very good for s, but E is very good for s.

Lemma 7.2.19 *Suppose C, D are contexts, s, t are terms , γ is an occurrence and $m \in N$.*

- *If C is very good for s, then C/γ is very good for s/γ for any occurrence γ.*
- *If C is a very good context relative to s, C/γ is a λ-hole, and t is an L-term, then $C\{\gamma \leftarrow t\}$ is a very good context relative to $s\{\gamma \leftarrow t\}$.*
- *If $K_c(C) = 0$ then C is very good for s provided C is a context for s.*
- *If $K_c(C) = 0$, C is very good for s, D is very good for t, $s \cdot t$ ($s \circ t$) is L-term, then $C \cdot D$($C \circ D$) is very good for $s \cdot t$ ($s \circ t$).*
- *If C is very good for s and $C/\gamma(\gamma \neq nil)$ is a hole, then s is an L-term.*
- *If C is very good for s, then C_m is very good for s for any $m \in N$.*
- *If C is very good for L-term s and t is a W-term, then $C \circ_c \Box_m$ is very good for $s \circ t$.*

Lemma 7.2.20 *Let C, D be contexts, s, t be terms and γ be an occurrence.*

1. *If C is a very good context relative to s, then $C\{\gamma \leftarrow \Box_m\}$ is very good for s.*
2. *If C is a very good context relative to s, C/γ is a $\lambda - hole$, and D is very good for L-term t, then $C\{\gamma \leftarrow D\}$ is very good for $s\{\gamma \leftarrow t\}$.*

Lots of cases are checked to prove this lemma. The following facts are also needed:

Lemma 7.2.21 *Suppose C, D are contexts, s, t are terms, u, w are tuples of terms and γ is an occurrence.*

1. *If C/γ is a hole, then $(s\{\gamma \leftarrow t\}/\gamma) = t$.*
2. *$C\{\gamma \leftarrow D\}[u] = (C[u]\{\gamma \leftarrow D[u]\})$ if C/γ is a hole.*
3. *If $D_1[u] = D_2[v]$, then $C\{\gamma \leftarrow D_1\}[u] = C\{\gamma \leftarrow D_2\}[v]$.*
4. *If $s/\gamma = t$, then $s\{\gamma \leftarrow t\} = s$.*
5. *If $C[u] = C[v]$ and $D[v] = C[u]/\gamma$, then $C[u] = C\{\gamma \leftarrow D\}[v]$.*
6. *If C/γ is a hole and $n \geq N_c(C)$, then $C[u]\{\gamma \leftarrow s\} = (C\{\gamma \leftarrow \Box_n\})[u@s]$.*
7. *If C/γ is a hole, $s \to t$, then $C[u]\{\gamma \leftarrow s\} \to C[u]\{\gamma \leftarrow t\}$*
8. *If C/γ is a hole and $n \geq N_c(C)$, then there exists $k \leq n$ such that $C[u]/\gamma = Proj(n, k, u)$.*
9. *If $Proj(n, k, u\tilde{o}w)$ is an L-term, then $Proj(n, k, u\tilde{o}w) = u_k \circ w_k$.*
10. *If C/γ is a hole, $n \geq N_c(C)$ and $C[u\tilde{o}w]/\gamma$ is an L-term, then $C[u\tilde{o}w]/\gamma = C[u]/\gamma \circ C[w]/\gamma$.*

Lemma 7.2.22 *Let C, D be contexts, then we have*

1. *$Lg(C\{\gamma \leftarrow D\}) \leq Lg(C) + Lg(D)$.*
2. *$Lg(C\{\gamma \leftarrow D\}) = Lg(C) + Lg(D)$ if C/γ is a hole.*
3. *$N_c(C\{\gamma \leftarrow D\}) \leq Max(N_c(C), N_c(D))$.*
4. *$N_c(C) = N_c(C_q)$.*
5. *$N_c(C) \geq 1$ if and only if there exists γ such that C/γ is a hole.*

where $Lg(C)$ is the length of the context C defined as follows:

$$Lg(\Box_m) = 0 \qquad Lg(id) = 1 \qquad Lg(v_c) = 1 \qquad Lg(\uparrow) = 1$$
$$Lg(\lambda_c(C)) = Lg(C) + 1 \qquad Lg(C \cdot D) = Lg(C) + Lg(D) + 1$$
$$Lg(C \circ_c D) = Lg(C) + Lg(D) + 1$$

7.3 Reduction of contexts

The main theorem in this section, the Preservation Theorem, basically says, the reduct of an increment of a strongly normalising term is still an increment of a strongly normalising term, and is smaller in some sense. Therefore we are interested in the contexts such that when we reduce a term $s = C[u]$ to $t = D[v]$, how the contexts C, D are related. We will define a notion of context reduction such that $C \to D$ if $C[u] \to D[u]$.

The contexts reduction is defined as:

$$
\begin{array}{l}
\mathit{ContReduc} : (\mathcal{C};\mathcal{C})\mathit{Set} \\
\quad id \circ_c D \to D \\
\quad v_c \circ_c id \to v_c \\
\quad v_c \circ_c (C \cdot D) \to C \\
\quad \uparrow \circ_c id \to \uparrow \\
\quad \uparrow \circ_c (C \cdot D) \to D \\
\quad \lambda_c(C) \circ_c D \to \lambda_c(C \circ_c (v_c \cdot (D \circ_c \uparrow))) \\
\quad (C \cdot D) \circ_c E \to (C \cdot E) \circ_c (D \cdot E) \\
\quad (C \circ_c D) \circ_c E \to C \circ_c (D \circ_c E) \\
\quad \lambda_c(C) \to \lambda_c(D) \text{ if } C \to D \\
\quad C \cdot D \to C' \cdot D \text{ if } C \to C' \\
\quad C \cdot D \to C \cdot D' \text{ if } D \to d' \\
\quad C \circ_c D \to C' \circ D \text{ if } C \to C' \\
\quad C \circ_c D \to C \circ D' \text{ if } D \to D'
\end{array}
$$

We have used $C \to D$ to denote $\mathit{ContReduc}(C, D)$.

It is defined by case analysis such that the following lemma holds:

Lemma 7.3.1 *If $C \to D$, then $C[u] \to D[u]$.*

Lemma 7.3.2 *Suppose that C, D are contests, γ is an occurrence.*

1. *If $C \to D$ and D/γ is a hole, then there exists δ such that C/δ is a hole.*

2. *If $C \to D$ and $K_c(C) = 0$, then $K_c(D) = 0$.*

Lemma 7.3.3 *Let s' be the result of the good inflation (C, w) of $s = C[u]$. Let D be a context such that $C \to D$, and let $t = D[u]$ (hence, $s \to t$ and $s' \to t'$). Then there exists a good inflation (D', w') of t whose result is t'.*

To make it explicit in ALF:

$$\frac{GoodInflation(s, C, u, w); C \to D}{\exists D' \exists m \exists v \exists w' (GoodInflation(D[u], D', m, v, w') \wedge I(D[u\tilde{\circ}w], D'[v\tilde{\circ}w']))}$$

This is proved by induction on $Lg(C)$.

Proof:

Let's see how the case $C = A \circ_c B$ is proved. Let $s = C[u] = A[u] \circ B[u] = a \circ b, \quad m = |u|$.

When $C = A \circ_c B \to D$, there are three possibilities according to the position of the redex.

1. The redex is within A. Suppose that $A \to E$, by I.H. there exists (E', u', w') which is the good inflation of $E[u]$, and $E'[u'\tilde{\circ}w'] = E[u\tilde{\circ}w]$. There are two cases: $K_c(A) = 0$ or $K_c(A) \geq 1$.

   (a) $K_c(A) = 0$. We have the following facts:

   - If $C \to D$ and $K_c(C) = 0$, then $K_c(D) = 0$. Hence $K_c(E) = 0$.
   - $(E \circ_c B_q, u'@u, w'@w)$ is a good inflation $E[u] \circ B[u]$ by Lemma 7.2.14 and Lemma 7.2.11, where $q = |u'|$.
   - The equality is checked easily by Lemma 7.2.10 and Lemma 7.2.2:

   $$\begin{aligned} & (E \circ_c B_q)[(u'@u)\tilde{\circ}(w'@w)] \\ = \;& (E \circ_c B_q)[(u'\tilde{\circ}w')@(u\tilde{\circ}w)] \\ = \;& (E[u'\tilde{\circ}w']) \circ (B[u\tilde{\circ}w]) \\ = \;& (E[u\tilde{\circ}w]) \circ (B[u\tilde{\circ}w]) \\ = \;& (E \circ_c B)[u\tilde{\circ}w] \end{aligned}$$

   (b) $K_c(A) \geq 1$. We have the following facts:

   - B is a hole and $B[u]$ is a W-term by definition.
   - $(E' \circ_c \Box_{m+1}, u'@B[u], w'@B[w])$ is a good inflation of $E[u] \circ B[u]$ by Lemma 7.2.14 and Lemma 7.2.11.

- The equality can be checked by Lemma 7.2.10:

$$\begin{aligned}
&(E' \circ_c \Box_{m+1})[(u'@B[u])\tilde{\circ}(w'@B[w])] \\
&= (E'[u'\tilde{\circ}w']) \circ (B[u]\tilde{\circ}B[w]) \\
&= (E[u\tilde{\circ}w]) \circ (B[u\tilde{\circ}w]) \\
&= (E \circ_c B)[u\tilde{\circ}w]
\end{aligned}$$

2. The redex is within B. Suppose $B \to E$. We have the following facts:

   - If $C \to D$, then C is not a hole. Therefore B is not a hole.
   - If $C \circ_c D$ is good for $a \circ b$ and D is not a hole, then $K_c(C) = 0$. Hence $K_c(A) = 0$ here as $A \circ_c B$ is good and B is not a hole.

   By I.H. there is a good inflation (E', u', w') of b such that $b' = E'[u'\tilde{\circ}w'] = E[u\tilde{\circ}w]$. From Lemma 7.2.14 $A \circ_c E'_m$ is good for $s = A[u] \circ E[u]$.

$$\begin{aligned}
&(A \circ_c E'_m)[(u@u')\tilde{\circ}(w@w')] \\
&= (A[u\tilde{\circ}w]) \circ (E'[u'\tilde{\circ}w']) \\
&= (A[u\tilde{\circ}w]) \circ (E[u\tilde{\circ}w]) \\
&= (A \circ_c E)[u\tilde{\circ}w]
\end{aligned}$$

   Therefore $(A \circ_c E'_m, u@u', w@w')$ is the good inflation of $(A \circ_c E)[u]$ in the lemma.

3. The redex is $A \circ_c B$. We argue according to the rule:

   (Ass): $C = (E \circ_c F) \circ_c B$. Two cases arises:

   (a) If $K_c(E) = 0$. We can prove that $(E \circ_c F) \circ_c B$ is good for $(a \circ b) \circ c$ implies that $E \circ_c (F \circ B)$ is good for $a \circ (b \circ c)$ (see Lemma 7.2.14). Thus $(E \circ_c (F \circ_c B), u, w)$ is the good inflation.

   (b) If $K_c(E) \geq 1$, then $F[u] \circ B[u]$ is a W-term, and $(E \circ_c \Box_{m+1}, u@(F[u] \circ B[u]), w@(F[w] \circ B[w]))$ is the good inflation.

   (Abs): $C = (\lambda_c(E)) \circ_c B$ and $D = \lambda_c(E \circ_c (v_c \cdot (B \circ_c \uparrow)))$. Two cases arise:

   (a) If $K_c(E) = 0$, $(\lambda_c(E \circ_c (v_c \cdot B \circ_c \Box_{m+1})), u@\uparrow, w@\uparrow)$ is the good inflation.

(b) If $K_c(E) \geq 1$, $(\lambda_c(E \circ_c \Box_{m+1}), u@(1 \cdot (B[u]\circ \uparrow)), w@(1 \cdot (B[w]\circ \uparrow))$ is the good inflation.

(Map): $C = (E \cdot F) \circ_c B$ and $D = (E \circ_c B) \cdot (F \circ_c B)$. $((E \circ_c B) \cdot (F \circ_c B), u, w)$ is the good inflation, because it can be proved that if $(E \cdot F) \circ_c B$ is good for $(a \cdot b) \circ c$ then $(E \circ_c B) \cdot (F \circ_c B)$. $((E \circ_c B)$ is good for $(a \circ c) \cdot (b \circ c)$ (see Lemma 7.2.14) and the equality is checked easily.

$\Box$

Lemma 7.3.4 *If (C, u, w) is a good inflation of s with result s', then there exists a very good inflation (C', u', w') of s with result s' that $K_{C'} \leq K_C$.*

This is proved by induction on the structure of C. The condition $K_{C'} \leq K_C$ is crucial to ensure that induction can be conducted.

Proof:

1. $C = \lambda_c(A)$. By I.H. there exists a very good inflation (A', u', w') for $s = A[u]$. Then we simply choose $(\lambda_c(A'), u', w')$.

2. $C = A \circ_c B$. Suppose that (A', u_A, w_B) and (B', u_B, w_B) are the very good inflations for $A[u]$ and $B[u]$ respectively, and $A'[u_A \tilde{\circ} w_A] = A[u \tilde{\circ} w]$ and $B'[u_B \tilde{\circ} w_B] = B[u \tilde{\circ} w]$. Again two cases arise:

   (a) $(A \circ_c B)[u]$ is a W-term, if there is no hole in $A \circ_c B$, then($A \circ_c B, u, w)$ is the very good inflation, otherwise $(\Box_1, (A \circ_c B)[u], (A \circ_c B)[u \tilde{\circ} w])$ will be the very good inflation. By lemma 7.2.19, for W-term $C[u]$, we always have $C[u] = C[u \tilde{\circ} w]$.

   (b) $(A \circ_c B)[u]$ is an L-term.

      - If $K_c(A') = 0$, then $A' \circ B'_q$ is very good for $A[u] \circ B[u]$ by 7.2.19, and $(A' \circ B'_q, u_A @ u_B, w_A @ w_B)$ is a good inflation for $A[u] \circ B[u]$, where $q = |u_A|$. The equality can be checked: $(A' \circ B'_q)[(u_A @ u_B) \tilde{\circ} (w_A @ w_B)] = (A' \circ B'_q)[(u_A \tilde{\circ} w_A) @ (u_B \tilde{\circ} w_B)] = A'[u_A \tilde{\circ} w_A] \circ B'[u_B \tilde{\circ} w_B] = A[u \tilde{\circ} w] \circ B[u \tilde{\circ} w]$. $K_c(A' \circ_c B'_q) \leq K_c(A \circ B)$ because $K_c(B'_q) = K_c(B')$.
      - $K_c(A') \geq 1$. Then $K_c(A) \geq 1$, and B will be a w-hole, this also implies that $A[u]$ will be an L-term because $A \circ_c B$ is very good. $A \circ_c$

$\Box_{q+1}$ will be a very good context for $A[u] \circ B[u]$ by lemma 7.2.19. $(A \circ_c \Box_{q+1}, u_A @ B[u], w_A @ B[w])$ will be the very good inflation. The equality is checked easily. $K_c(A' \circ_c \Box_{q+1}) = K_c(A') + 1 \leq K_c(A \circ_c B)$ because $K_C(B) = 1$.

3. $C = A \cdot B$. Similarly to the above case.

□

Combining the last two lemmas, we get the following lemma which is needed in Lemma 7.3.8:

Lemma 7.3.5 *Suppose that s' is the result of a very good inflation (C, u, w) of $s = C[u]$, $C \rightarrow D$, and $t' = D[u\tilde{o}w]$, then there exists a very good inflation (D', u', w') of $t = D[u]$ such that $D'[u'\tilde{o}w'] = D[u\tilde{o}w]$.*

This is very direct from Lemma 7.3.3 and lemma 7.3.4, and the ALF proof is not big, but it took very long time to check. If we look at these lemmas in ALF, they use several existential quantifiers.

Lemma 7.3.3 is encoded as follows in ALF:

$$\frac{GoodInflation(C, u, w); C \rightarrow D}{EX_4(A, m, u', w')(And(GoodInflation(A, u', w'), I(D[u\tilde{o}w], A[u'\tilde{o}w'])))}$$

Lemma 7.3.4 looks like :

$$\frac{GoodInflation(C, u, w)}{EX_4(A, m, u', w')AND_3(VeryGoodInflation(C[u], A, u, w'), I(C[u\tilde{o}w], A[u'\tilde{o}w']), K_c(C) \geq K_c(A))}$$

Lemma 7.3.5 is

$$\frac{GoodInflation(C, u, w); C \rightarrow D}{EX_4(A, m, u', w')AND_3(VeryGoodInflation(D[u], A, u, w'), I(D[u\tilde{o}w], A[u'\tilde{o}w']), K_c(D) \geq K_c(A))}$$

where

$$VeryGoodInflation(C[u], A, u, w') = GoodInflation(C[u], A, u, w') \wedge VeryGood(A, C[u])$$

I is the intensional equality, EX_4 denotes four successive existential quantifiers and AND_3 denotes conjunction of three sets.

Now let's see what is happening when a term $s = C[u]$ is reduced. Intuitively there are three possibilities: the redex is in C, the redex is in some u_k or there is an interaction between C and u. For our purpose and simplicity, we only consider the reduction $C[u\tilde{o}w] \rightarrow t$. In this case, three cases arise according to the occurrence of the redex:

1. The reduction takes place in the context, i.e. $C \twoheadrightarrow D$ and $t = D[u\tilde{\circ}w]$.

2. The redex is in some hole $\Box_k$, $(u \circ w)_k \twoheadrightarrow r$ and $t = C'[(u\tilde{\circ}w)@r]$ where C' is the context by replacing the square $\Box_k$, where u_k is reduced to r, by $\Box_{N_c(C)+1}$.

3. There exists interaction between C and $u\tilde{\circ}w$. It is not easy to state this case precisely. There are three cases which may cause interaction:

   (a) $(\Box_m \circ_c D)[u\tilde{\circ}w] \twoheadrightarrow b$.

   (b) $(v_c \circ_c \Box_m)[u\tilde{\circ}w] \twoheadrightarrow b$.

   (c) $(\uparrow \circ_c \Box_m)[u\tilde{\circ}w] \twoheadrightarrow b$

   In the case C is a very good context of $C[u]$, 3b and 3c become to case 2. The case 3a happens only when the rule (Ass.) is applied, and $(u\tilde{\circ}w)_m = u_m \circ w_m$. This means there exists a very good inflation for $C[u]$ and the result is still $C[u\tilde{\circ}w]$, which is what we want for proving the Preservation Theorem.

Lemma 7.3.6 *Suppose that (C, u, w) is a very good inflation and $C[u\tilde{\circ}w] \twoheadrightarrow b$, then one of the following holds:*

1. *There exists a context D such that $C \twoheadrightarrow D$ and $b = D[u\tilde{\circ}w]$.*

2. *There exist $\gamma \in \mathcal{L}, c \in \Lambda\sigma_0$ such that C/γ is a hole, and $C[u\tilde{\circ}w]/\gamma \twoheadrightarrow c$ and $b = (C\{\gamma \leftarrow \Box_q\})[u@c]$.*

3. *There exist $D \in \mathcal{C}, n \in N, u', w' \in \Lambda\sigma_0^n$ such that (D, u', w') is a very good inflation of $C[u]$, $b = D[u'\tilde{\circ}w']$, $Lg(D) > Lg(C)$ and $K_c(D) \leq K_c(C)$.*

Remark 7.3.7 *The second statement has been changed several times in order to present all the information when this lemma is applied. What we should present is the most original information which can derive other information when it is needed. In this case, the most original information is presented in terms of the "position".*

The first statement is:

There exist $\gamma \in \mathcal{L}, k \in N, c \in \Lambda\sigma_0$ such that $(u\tilde{\circ}w)_k \twoheadrightarrow c$ and $b = (C\{\gamma \leftarrow \Box_q\})[u@c]$.

Then one need the information that C/γ is a hole, and $k \leq n = |u|$ and $C[u\tilde{\circ}w]/\gamma = (u\tilde{\circ}w)_k$, which we have when the lemma is proved. However, it is not enough still.

One need to say that $C[u]/\gamma = u_k$ and $C[w]/\gamma = w_k$, which we can not get from the revised statement. In fact all the information is stored in the following statement:

There exist $\gamma \in \mathcal{L}, c \in \Lambda\sigma_0$ such that C/γ is a hole, and $C[u\tilde{\circ}w]/\gamma \twoheadrightarrow c$ and $b = (C\{\gamma \leftarrow \Box_q\})[u@c]$.

Proof:

This lemma is proved by induction on the structure of C. Let's see how the lemma is proved when $C = A \circ_c B$. We prove this case by analysing the rule $(A \circ_c B)[u\tilde{\circ}w]$. We shall use the notation $C(n,\gamma) = C\{\gamma \leftarrow \Box\}$.

1. The redex is in $A[u\tilde{\circ}w]$ and $A[u\tilde{\circ}w] \twoheadrightarrow b$. Three cases arise according to the I.H.:

   (a) There exists a context A' such that $A \twoheadrightarrow A'$. Therefore $A \circ_c B \twoheadrightarrow A' \circ_c B$ and $b = (A' \circ_c B)[u\tilde{\circ}w]$.

   (b) A/γ is a hole and $A[u]/\gamma \twoheadrightarrow a'$ and $b = A(n,\gamma)[(u\tilde{\circ}w)@a']$. Then $A \circ_c B/1\gamma$ is a hole, $(A \circ_c B)[u]/1\gamma \twoheadrightarrow a' \circ B[u]$ and $b \circ B[u\tilde{\circ}w] = ((A \circ_c B)\{1\gamma \leftarrow \Box_k\})[(u\tilde{\circ}w)@a']$.

   (c) There exists a very good inflation $(A[u], A', u', w')$ with the result $b = A'[u'\tilde{\circ}w']$. Two cases arise:

      i. $K_c(A') = 0$. If $K_c(B) = 0$, then we take the inflation $(A' \circ_c B, u, w)$. The result of the inflation is $(A' \circ_c B)[u'\tilde{\circ}w'] = b \circ B[u\tilde{\circ}w]$ because $K_c(A') = 0$, $A'[u'\tilde{\circ}w'] = A'[u\tilde{\circ}w]$. Otherwise, we take the inflation $(A' \circ_c B_q, u'@u, w'@w)$. The result is $b \circ B[u\tilde{\circ}w]$. In both cases they are very good inflations by Lemma 7.2.19 with the same result $b \circ B[u\tilde{\circ}w]$.

      ii. $K_c(A') \geq 1$. We take the inflation $(A' \circ_c \Box_q, u'@B[u], w'@B[w])$. In fact, in this case, $K_c(A) \geq 1$, hence B is W-hole. By Lemma 7.2.19, it is a very good inflation. The result is $b \circ B[u\tilde{\circ}w]$.

   For both cases the two equalities are true.

2. The redex is in $B[u\tilde{\circ}w]$, it is similar to the above case.

3. The redex is $(A \circ_c B)[u\tilde{\circ}w]$.

(IdL_c)**:** $C = id \circ_c B \to B$, $(id \circ_c B)[u\tilde{\circ}w] \to B[u\tilde{\circ}w]$. Hence the first case of the lemma holds.

$(VrId_c)$**:** $C = v_c \circ_c id \to v_c$. The first case of the lemma holds.

$(VrCons_c)$**:** $C = v_c \circ_c (A \cdot B)$ and $v_c \circ_c (A \cdot B) \to A$. The first case of the lemma holds.

$(v_c \circ_c \Box_k)$**:** $C = v_c \circ_c \Box_k$. In this case we have the following facts:

- If $C \circ_c D$ is very good for $a \circ b$ and $K_c(C \circ_c D) \geq 1$, then $a \circ b$ is an L-term; hence u_k is an L-term because $v_c \circ_c \Box_k$ is very good for $1 \circ u_k$.
- $(v_c \circ_c \Box_k)[u\tilde{\circ}w] = 1 \circ (u_k \circ w_k)$ because u_k is an L-term and hence $(u\tilde{\circ}w)_k = u_k \circ w_k$.
- If $1 \circ (a \circ b) \to c$, then $c = 1 \circ d$ where $a \circ b \to d$. This concludes that the redex is in $(u\tilde{\circ}w)_k = u_k \circ w_k$.

Therefore the second case of the lemma holds, $\gamma = 2$ and $1 \circ (u_k \circ w_k)/2 \to c$.

$(\uparrow \circ_c \Box_k)$**:** $C = \uparrow \circ \Box_k$, it is the same as the case above.

$(ShId_c)$**:** $C = \uparrow \circ_c id \to \uparrow$. The first case of the lemma holds.

(ShCons): $C = \uparrow \circ_c (A \cdot B) \to B$. The first case of the lemma holds.

(Interaction): $C = \Box_k \circ_c B$. The following facts were proved:

- B is a W-hole and u_k is an L-term because $\Box_k \circ_c B$ is very good.
- $b = u_k \circ (w_k \circ B[u\tilde{\circ}w])$.
- $(u_k \circ_c \Box_{n+1}, u@B[u], w@(w_k \circ B[w]))$ is a good inflation.
- $u_k \circ_c \Box_{n+1}$ is a very good context for $(u_k \circ_c \Box_{n+1})[u]$.
- $b = (u_k \circ_c \Box_{n+1})[u\tilde{\circ}w]$.
- $Lg(u_k \circ_c \Box_{n+1}) = Lg(u_k) + 1 > 1 = Lg(\Box_k \circ_c B)$.
- $K_c(u_k \circ_c \Box_{n+1}) = 1 < 2 = \Box_k \circ_c B$.

All these facts mean that the third case of the lemma holds.

(Abs): $C = \lambda_c(A) \circ_c B \to \lambda_c(A \circ_c (v_c \cdot (B \circ_c \uparrow)))$. The first case of the lemma holds.

(Map): $C = (A \cdot B) \circ_c E \to (A \circ_c E) \cdot (B \circ_c E)$. The first case of the lemma holds.

(Ass_c)**:** $C = (A \circ_c B) \circ_c E \to A \circ_c (B \circ_c E)$. The first case of the lemma holds.

$\square$

The following lemma will enable us to use induction on the triple $\theta_{s,s'} = (dp(s), lg(s) - lg(C), \Sigma\mu_k dp(w_k))$ to prove the Preservation Theorem.

Lemma 7.3.8 *If s' is the result of a very good inflation (C, u, w) of some term s, and $s' \to t'$, then one of the following holds:*

- *there exists a term t such that $s \to t$ and t' is the result of a very good inflation of t.*
- *t' is the result of a very good inflation (D, a, b) of s and $lg(D) > lg(C)$.*
- *there exists some term r such that $w_k \to r$ and t' is the result of the very good inflation $(C', u@u_k, w@r)$ of s where $C' = C\{\gamma \leftarrow \square_n\}$ and $n = Lg(u) + 1$.*

Proof:

This is proved by analysing the position of the redex of s' based on Lemma 7.3.6.

We will prove that for any t' such that $s' = C[u\tilde{\circ}w] \to t'$, t' can be the result of a very good inflation of some t such that $\theta_{t,t'} < \theta_{s,s'}$, so the I.H. can be applied.

There are three cases by Lemma 7.3.6:

1. The reduction takes place in the context.

   By Lemma 7.3.5, $t = D[u]$, $s \to t$ and there exists a very good inflation (D', u', w') of t and the result is still t'.

2. The reduction takes place in some hole of the context C, i.e. there exists $\gamma \in \mathcal{L}, k \in N$ such that $C[u\tilde{\circ}w]/\gamma = (u\tilde{\circ}w)_k$ and $(u\tilde{\circ}w)_k \to t'$. Let $C' = C\{\gamma \leftarrow \square_n\}$. Two cases arise: $(u\tilde{\circ}w)_k$ is either an L-term or a W-term.

   (a) $(u\tilde{\circ}w)_k$ is an L-term. In this case, $(u\tilde{\circ}w)_k = u_k \circ w_k$ and u_k must be an L-term. We argue according to the redex. Five cases arise:

   (Ass): Let $u_k = a \circ b$, and therefore, $s = C'[u@(a \circ b)]$ and $s' = C'[(u\tilde{\circ}w)@((a \circ b) \circ w_k)]$. Hence $t' = C'[(u\tilde{\circ}w)@(a \circ (b \circ w_k))]$, where $C' = C\{\gamma \leftarrow \square_n\}$. Let $C'' = C\{\gamma \leftarrow a \circ_c \square_n\}$ and consider the inflation $(C[u], C'', u@a, w@w_n)$, where $w_n = IfThEl(b, b \circ w_k, w_k)$, which is a W-term by Lemma 7.2.10. The following facts are proved:

- $C[u]\{\gamma \leftarrow (a \circ_c \Box_n)[u@b]\} = C[u]$ by Lemma 7.2.21.
- C'' is a very good context of $C[u]$ by Lemma 7.2.20.
- $(C[u], C'', u@b, w@w_k)$ is a very good inflation.
- $t' = C'[(u\tilde{\circ}w)@(a \circ (b \circ w_k))] = C''[((u@b)\tilde{\circ}(w@w_n))]$ by Lemma 7.2.21.
- $Lg(C'') > Lg(C)$ by Lemma 7.2.22.

Therefore the second case of the Lemma 7.3.8 holds.

(Abs): Let $u_k = \lambda(a)$, $C'' = C\{\gamma \leftarrow \lambda_c\Box_n\}$ and consider the inflation $(C[u], C'', u@a, w@w_n)$ where $w_n = IfThEl(a, a\circ(1\cdot(w_k\circ\uparrow)), 1\cdot(w_k\circ\uparrow))$, which is a W-term by Lemma 7.2.10. The following facts were proved:

- $C[u]\{\gamma \leftarrow (\lambda_c\Box_n)[u@a]\} = C[u]$.
- C'' is good for $C[u]$ be Lemma 7.2.15.
- C'' is very good for $C[u]$ by lemma 7.2.20.
- $Lg(C'') > Lg(C)$ by Lemma 7.2.22.

These facts prove that the second case of Lemma 7.3.8 holds.

(Map): Let $u_k = a \cdot b$, $C'' = \{\gamma \leftarrow \Box_n \cdot \Box_{n+1}\}$ and let us consider the inflation $I = (C[u], C'', u@<a,b>, w@<w_n, w_{n+1}>)$, where $w_n = IfThEl(a, a \circ w_k, w_k)$, $w_{n+1} = IfThEl(b, b \circ w_k, w_k)$. We have the following facts:

- $C[u]\{\gamma \leftarrow a \circ b\} = C[u]$.
- C'' is good for $C[u]$ by Lemma 7.2.15.
- C'' is very good for $C[u]$ by Lemma 7.2.20.
- $(C[u], C'', u@<a,b>, w@<w_n, w_{n+1}>)$ is a very good inflation.
- $C''[(u@<a,b>)\tilde{\circ}(w@<w_n, w_{n+1}>)] = C'[(u\tilde{\circ}w)@((a \circ w_k)\cdot(b \circ w_k))] = t'$.
- $Lg(C'') > Lg(C)$ by Lemma 7.2.22.

These facts suggest that the second case of the lemma holds.

(ComL): The redex is contained in u_k, and $u_k \rightarrow a$. Let $t = C'[u@a]$. By Lemma 7.2.21 $s \rightarrow t$, and C' is a good context for t by Lemma 7.2.15.

i. If a is an L-term. $(C\{\gamma \leftarrow \Box_n\}, u@a, w@w_k)$ is a very good inflation with the result $(C\{\gamma \leftarrow \Box_n\})[(u\tilde{\circ}w)@(a \circ w_k)]$.

ii. If a is a W-term, $C\{\gamma \leftarrow \Box_n\}$ is a good context for $t = (C\{\gamma \leftarrow \Box_n\})[u@a]$, hence $(C\{\gamma \leftarrow \Box_n\}, u@a, w@(a \circ w_k)$ is still a good inflation. By Lemma 7.3.4 there exists a very good inflation with the same result. Therefore the first case of the Lemma holds.

(ComR): In this case, $w_k \to b$, and $(C[u], C', u@u_k, w@b)$ is the very good inflation with the result $C'[(u\tilde{\circ}w)@(u_k \circ b)]$. The third case of the lemma holds.

(b) $(u\tilde{\circ}w)_k$ is a W-term, hence u_k is also a W-term. In this case $(u\tilde{\circ}w)_k = w_k$, $w_k \to b$, and b must be a W-term. By Lemma 7.2.20 $C\{\gamma \leftarrow \Box_n\}$ is very good for $C[u](\gamma, b)$. Therefore the inflation $(C\{\gamma \leftarrow \Box_n\}, u@u_k, w@b)$ is very good, and the result is $(C\{\gamma \leftarrow \Box_n\})[(u\tilde{\circ}w)@w_k]$.

3. For the third case of Lemma 7.3.6, the second case of the lemma holds.

□

Now we are in the position to prove the main theorem:

Theorem 7.3.9 (Preservation) *Let $s \in SN$ and let s' be the result of a very good inflation (C, w) of $s = C[u]$. Then $s' \in SN$.*

Proof:

This is proved by induction over a triple $\theta_{s,s'} = (dp(s), lg(s) - lg(C), \Sigma\mu_k dp(w_k))$ where $\mu_k = card\{\gamma : C/\gamma = \Box_k\}$, i.e. the number of occurrences of $\Box_k$ in C.

By induction on the triple $\theta_{s,s'} = (dp(s), lg(s) - lg(C), \Sigma\mu_k dp(w_k))$. By Lemma 7.3.8, there are three case. For the first case, b is a very good inflation of t and the first component decreases and t is strongly normalising, therefore the I.H. applies. For the second case, b is the result of a very good inflation of s and the second component decreases, hence the I.H. applies. For the third case, b is a the result of a very good inflation of s and the third component decreases, so the I.H. applies.

□

Chapter 8

Proving SN of σ in ALF

In this chapter we shall present the strong normalisation proofs of σ_0 and σ in ALF.

8.1 Strong Normalisation of σ_0

The strong normalisation of σ_0 is proved by the elimination rule of $\Lambda\sigma_0$ in section 6.2. Therefore we need to give the proof objects $e_1, \cdots, e_6$ when C is SN. It is easy to get the proofs e_1, e_2 and e_3. Since no rules of the σ_0-calculus contains 'λ' or '$\cdot$' as head symbol, the proofs e_4 and e_5 are also easy. The difficulty comes out when coming to the proof e_6:

$$e_6 : (a, b : \Lambda\sigma_0; SN(a); SN(b))SN(a \circ b)$$

or more precisely when coming to the rule:

$$(Abs) : \quad (\lambda s) \circ t \rightarrow \lambda(s \circ (1 \cdot (t \circ \uparrow)))$$

If we can prove that

$$(*) \quad \text{If } s \in SN \text{ then } s \circ \uparrow \in SN$$

then we can use induction on $(dpth(s), lgth(s), dpth(t), lgth(t))$ to get the proof object e_6. To solve the problem $(*)$, we have introduced a context calculus in last chapter and finally give the proof of $(*)$ by the Preservation Theorem.

However, it is easy to prove that reduction which does not involve the rule (Abs) is strongly normalising.

Lemma 8.1.1 *Let $s, t \in \Lambda\sigma_0$.*

1. *$\lambda(t)$ are strongly normalising if and only if t is strongly normalising.*
2. *$s \cdot t$ is strongly normalising if and only if s and t are strongly normalising.*
3. *If $s \circ t$ is strongly normalising, then s and t are strongly normalising.*
4. *Suppose that s, t are W-terms. If s and t are strongly normalising, then $s \circ t$ is strongly normalising.*
5. *If $t \circ \uparrow$ is strongly normalising whenever t is, then $s \circ t$ is strongly normalising for any strongly normalising terms s and t.*
6. *Any W-term is strongly normalising.*

In ALF, this lemma was proved by induction on depth and length. We shall give the details of the induction proof in chapter 9 when proving termination of the calculus s.

Now we come to the lemma where the Preservation Theorem is used and the problem $(*)$ is solved:

Lemma 8.1.2 *Let s be a L-term, then $s \circ \uparrow$ is the result of the very good inflation $(\Box_1, \uparrow)$ of s. Therefore, $s \circ \uparrow$ is strongly normalising whenever L-term s is strongly normalising.*

It is easy to see that $s \circ \uparrow$ is the result of a very good inflation $(\Box_k, \uparrow)$ of s when s is not a W-term, and it is strongly normalising by the preservation theorem. If s is a W-term, then $s \circ \uparrow$ is also a W-term, and it is strongly normalising. Hence we have the following lemma:

Lemma 8.1.3 *If s is strongly normalising, then $s \circ \uparrow$ is strongly normalising.*

Now we have the proof object e_6:

Lemma 8.1.4 *If s, t are strongly normalising, then $s \circ t$ is strongly normalising.*

This is proved by induction on $(dpth(s), lgth(s), dpth(t), lgth(t))$.

Having got all the proof objects $e_1, \cdots, e_6$, we prove the strong normalisation of σ_0 by induction on the structure of σ_0-terms:

Theorem 8.1.5 *σ_0 is strongly normalising.*

8.2 Strong Normalisation of σ

Having proved that the calculus σ_0 is terminating, now we can prove that the calculus σ is terminating.

We have defined a translation F from σ to σ_0 in definition 2.2.2. By checking the function F is really a strict interpretation from σ to σ_0 and proposition 2.2.3, we get the strong normalisation of σ:

Theorem 8.2.1 *The calculus σ is strongly normalising.*

Lemma 3.1.5 is easily checked in ALF.

Chapter 9

Formalising λs and SN of s in ALF

In this chapter we implement the calculus λs and strong normalisation proofs of the calculus s given in section 3.2.

9.1 Implementation of the calculus λs in ALF

The set of λs-terms is given as:

$$\Lambda s ::= N \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s \sigma^i \Lambda s \mid \varphi^i_k \Lambda s$$

where $i \geq 1, k \geq 0$.

Λs is an inductively defined set in ALF with five constructors, which correspond to the five kinds of terms:

$$
\begin{aligned}
&\mathit{Sterm} : \mathit{Set}\\
&\quad \mathit{Var} : (n : N; n > 0)\mathit{Sterm}\\
&\quad \mathit{App} : (a, b : \mathit{Sterm})\mathit{Sterm}\\
&\quad \mathit{Lam} : (a : \mathit{Sterm})\mathit{Sterm}\\
&\quad \mathit{Sub} : (a : \mathit{Sterm}; j : N; j > 0; b : \mathit{Sterm})\mathit{Sterm}\\
&\quad \mathit{Rename} : (i, k : N; i > 0; a : \mathit{Sterm})\mathit{Sterm}
\end{aligned}
$$

where we use $Sub : (a : Sterm; j : N; j > 0; b : Sterm)$ to denote $a\sigma^j b$ and $Rename : (i, k : N; i > 0; a : Sterm)$ to denote $\varphi^i_k a$ in ALF.

This is known as the introduction rule of the set Λs.

In this thesis we are interested in the rules of s-calculus:

$$
\begin{array}{ll}
\sigma - \lambda - transition & (\lambda a)\sigma^i b \to \lambda(a\sigma^{i+1} b) \\
\sigma - app - transition & (a_1 a_2)\sigma^i b \to (a_1\sigma^i b)(a_2\sigma^i b) \\
\sigma - destruction & n\sigma^i b \to \begin{cases} n-1 & \text{if } n > i \\ \varphi^i_0 b & \text{if } n = i \\ n & \text{if } n < i \end{cases} \\
\varphi - \lambda - transition & \varphi^i_k(\lambda a) \to \lambda(\varphi^i_{k+1} a) \\
\varphi - app - transition & \varphi^i_k(a_1 a_2) \to (\varphi^i_k a_1)(\varphi^i_k a_2) \\
\varphi - destruction & \varphi^i_k n \to \begin{cases} n+i-1 & \text{if } n > k \\ n & \text{if } n \leq k \end{cases}
\end{array}
$$

This set of rules is defined in ALF as follows:

$SRules : (s, t : Sterm)Set$

$SubLamTran : (p : j > 0)SRules(\lambda(a)\sigma^j b, \lambda(a\sigma^{succ(j)} b)))$

$SubAppTran : (p : j > 0)SRules((a_1, a_2)\sigma^j b, (a_1\sigma^j b)(a_2\sigma^j b))$

$SubDes_1 : (p : n > i)SRules(Var(succ(n))\sigma^{succ(i)} b, Var(n))$

$SubDes_2 : SRules(Var(succ(n))\sigma^{succ(n)} b, \varphi_0^{succ(n)} b)$

$SubDes_3 : (p : i > n)SRules(Var(succ(n))\sigma^{succ(i)} b, Var(succ(n)))$

$RenLamTran : (p : i > 0)SRules(\varphi^i_k(\lambda(a)), \lambda(\varphi^i_{succ(k)} a))$

$RenAppTran : (p : i > 0)SRules(\varphi^i_k(a_1 a_2), (\varphi^i_k a_1)(\varphi^i_k a_2))$

$RenDes_1 : SRules(\varphi_k^{succ(i)} Var(succ(plus(n, k))), Var(succ(plus(n, plus(i, k)))))$

$RenDes_2 : SRules(\varphi^{succ(i)}_{plus(k, succ(n))} Var(succ(n)), Var(succ(n)))$

where $succ(n) = n + 1$.

To improve readability we are using the notations introduced in chapter 3. This is called the introduction rule of the set of s-reduction rules, where the nine constructors correspond to the nine s-reduction rules.

We have hidden some arguments in the implementation of the set of s-reduction rules. To improve readability we shall hide some uninteresting arguments when their types can be derived in the expression. For instance, the type of a in the first constructor must be $Sterm$ and the type of j must be N by the type of $Sub(Lam(a), j, p, b)$.

Remark 9.1.1 *$succ(n)$ is definition-ally equal to $plus(n, 1)$ but not $plus(1, n)$. However, $succ(n)$ is equal to $plus(1, n)$ intensionally. We use $I(N, m, n)$ to denote m, n*

are intensionally equal. The set $I(A,a,b)$ is defined by the following rules:

Formation rule of I:

$$\frac{A : Set \quad a, b : A}{I(A,a,b) : Set}$$

Introduction rule of I:

$$\frac{a : A}{r(a) : I(A,a,a)}$$

When defining SN of s, we need the notion of one step reduction. This is defined by analysing which red-ex is reduced. Basically there are two cases, reducing the whole term as a redex and reducing a sub-term as a redex. The first case corresponds to use a rule directly to the term, and the second case corresponds to use a rule to a sub-term. One step reduction is defined as follows:

$S\_OneStep : (s,t : Sterm)Set$

$\quad RuleDirect : (SRules(s,t))S\_OneStep(s,t)$

$\quad App\_Congl : (S\_OneStep(s_1,s_2))S\_OneStep(s_1t, s_2t)$

$\quad App\_Congr : (S\_OneStep(s_1,s_2))S\_OneStep(ts_1, ts_2)$

$\quad Lam\_Cong : (S\_OneStep(s_1,s_2))S\_OneStep(\lambda(s_1), \lambda(s_2))$

$\quad Sub\_Congl : (S\_OneStep(s_1,s_2))S\_OneStep(s_1\sigma^{i+1}t, s_2\sigma^{i+1}t)$

$\quad Sub\_Congr : (S\_OneStep(s_1,s_2))S\_OneStep(t\sigma i + 1s_1, t\sigma^{i+1}s_2)$

$\quad Ren\_Cong : (S\_OneStep(s_1,s_2))S\_OneStep(\varphi_k^{i+1}s_1, \varphi_k^{i+1}s_2)$

To get the reduction relations $a \to_s^+ b$ and $a \to_s^\star b$, just replace the arguments A and R by *Sterm* and *S_OneStep* in *StepPlus* and *StepStar* respectively.

9.2 Proving SN of s directly in ALF

To avoid information redundancy we use SN_s to denote strong normalisation for the calculus λs:

$$SN_s \equiv [h]SN(Sterm, OneStep, h) : (h : Sterm)Set$$

Now let us see how to implement the strong normalisation proof of s given in section 3.2.3.

Theorem 3.2.3 is proved by induction on the structure of terms, or using elimination rule:

$$\begin{aligned}
&STerm\_elim : (C : (\Lambda s)Set) \\
&\quad (e_1 : C(Var(n))) \\
&\quad (e_2 : (a, b : \Lambda s; C(a); C(b))C(ab)) \\
&\quad (e_3 : (a : \Lambda s; C(a))C(\lambda a) \\
&\quad (e_4 : (a : \Lambda s; i, k : N; C(a))C(\varphi^i_k a) \\
&\quad (e_5 : (a, b : \Lambda s; i : N; C(a); C(b))C(a\sigma^i b) \\
&\quad (a : \Lambda s) \\
&\quad C(a)
\end{aligned}$$

To this end we need the following lemmas which will give the proof objects of e_1, e_2, e_3, e_4 and e_5 when C is SN_s:

Lemma 9.2.1 *$ab \in SN_s$ if and only if $a \in SN_s$ and $b \in SN_s$.*

Lemma 9.2.2 *$\lambda a \in SN_s$ if and only if $a \in SN_s$.*

Lemma 9.2.3 *For any $i \geq 1, k \geq 0$, $\varphi^i_k a \in SN_s$ if and only if $a \in SN_s$.*

Lemma 9.2.4 *For any $i \geq 1$, $a\sigma^i b \in SN_s$ if and only $a, b \in SN_s$*

Intuitively, SN_s holds for all normal forms because for them the premise of $SNintr$ is vacuously true. Especially every variable is strongly normalising, i.e. e_1 is easy to get.

After proving all these lemmas, we finish the proof of Theorem 9.3.12 by induction on the structures of terms.

Remark 9.2.5 *By using pattern matching, we do not need to write the elimination rule of Sterm. Theorem 3.2.3 reads in ALF: $SN_s : (a : \Lambda s)SN_s(a)$*

By pattern matching on the argument a, we get the following equations:

$$\begin{aligned}
&SN_s : (a : \Lambda s)SN_s(a) \\
&\quad SN_s(Var(n)) = ?_{e_1} \\
&\quad SN_s(ab) = ?_{e_2} \\
&\quad SN_s(\lambda(a)) = ?_{e_3} \\
&\quad SN_s(\varphi^i_k a) = ?_{e_4} \\
&\quad SN_s(a\sigma^i b) = ?_{e_5}
\end{aligned}$$

Here we have the same tasks to give those proof objects $e_1, \cdots, e_5$.

Lemma 9.2.1, 9.2.2 are proved by induction on the derivation sequences, that is by SN_s elimination.

To prove Lemma 9.2.3 and Lemma 9.2.4 we need combine the SN_s elimination and term elimination, which correspond to induction on (dpth(a), lgth(a))(Here we note that dpth(a) is the number of reductions in the longest derivation path starting from term a, however we don't need to formalise dpth(a), which is a partial function defined only on strongly normalising terms).

We shall give the ALF proof details of Lemma 9.2.2 and Lemma 9.2.3.

Proof of Lemma 9.2.2

It is easy to prove that the lemma holds in classical logic if using the definition "there are no infinite derivations". But when proving it in ALF, we can't use the the classical law of refutation. We must use introduction rules and elimination rules to give a constructive proof.

Let us first prove the "only if" part.

We define a predicate $P_1(a) \equiv \forall x \in \Lambda s((a = \lambda(x)) \to SN_s(x))$(We shall use $\to$ as imply when no confusion arises).

We will prove the following facts:

1. $\forall x \in \Lambda s(SN_s(x) \to P_1(x))$

2. $\forall x \in \Lambda s(P_1(\lambda(a)) \to SN_s(a))$

It is easy to see that $P_1(\lambda(a))$ implies $SN_s(a)$ by definition of P_1. This is proved in ALF by giving a function which for any proof of $P_1(\lambda(a))$ gives a proof of $SN_s(a)$:

Suppose we have a proof $h : P_1(\lambda(a))$; by the elimination rule of $\forall$, we have a proof $Forall\_elim(h, \lambda(a)) : (\lambda(a) = \lambda(a)) \to SN_s(a))$, then by the elimination rule of $\to$ and a proof $r : \lambda(a) = \lambda(a)$, we get a proof of $SN_s(a)$. The final proof of $P_1(\lambda(a)) \to SN_s(a)$ for any $a \in \Lambda s$ looks like:

$Imply\_intro([h]Imply\_elim(Forall\_elim(h, \lambda(a)), r)$

Using introduction rule of $\forall$, we get the proof of 2.

The main task is to prove that $\forall x \in \Lambda s(SN_s(x) \to P_1(x))$. This is proved by the induction principle $RecSN_s$, which amounts to induction on derivations. As we said, we should prove that $P_1(x)$ is closed under one step reduction, i.e.:

$$\frac{m : \Lambda s; h : (n : \Lambda s; : OneStep(m,n))P_1(n)}{? : P_1(m)}$$

We shall use

$$\frac{A}{? : B}$$

to denote that under the assumption A to find a proof of type B.

The problem $P_1(m)$ is solved by introduction rules of universal quantifier, imply and by finding a proof object of type $SN_s(x)$:

$$\frac{x : \Lambda s; m \equiv \lambda(x)}{? : SN_s(x)}$$

The problem $SN_s(x)$ is solved by the introduction rule of SN_s and finding a proof object of type $SN_s(b)$:

$$\frac{b : \Lambda s; h_2 : OneStep(x,b)}{? : SN_s(b)}$$

This is proved by the fact we just proved $P_1(\lambda(b)) \to SN_s(b)$ and a proof of $P_1(\lambda(b))$. The proof of $P_1(\lambda(b))$ is obtained from the proof $h : (n : \Lambda s; : OneStep(m,n))P_1(n)$, where $m = \lambda(x), n = \lambda(b)$ and $OneStep(m,n)$ because of $h_2 : OneStep(x,b)$.

The "if" direction of the lemma is proved in the same way as follows:

Suppose $P_2(x) \equiv SN_s(\lambda(x))$, then we can prove that $P_2(x)$ is closed under one step reduction. By reduction on derivations, we prove that

$\forall x \in \Lambda s(SN_s(x) \to P_2(x))$

It is easy to see that $P_2(x)$ implies $SN_s(\lambda(x))$.

□

Proof of Lemma 9.2.3

Let $P_4(x) \equiv \forall i,k \in N(SN_s(\varphi^i_k x))$. We prove that $\forall x \in \Lambda s(SN_s(x) \to P4(x))$. This is written in ALF as:

$$(x : \Lambda s; SN_s(x))P_4(x))$$

This is proved by induction on lexicographic order of $(dpth(x), lgth(x))$, which is first using induction principle RecSN and then induction on the structure of terms.

Let us first state this induction principle:

$InducDpthLgth : (P : (\Lambda s)Set$

$wf\_ih : (n : \Lambda s; SN_s(n); ih : (x : \Lambda s; Or(OneStep(n,x), SubTerm(x,n)))P(x))P(n)$

$n : \Lambda s$

$SN_s(n))P(n)$

where $SubTerm(x, n)$, denotes x is a proper subterm of n, is defined as:

$$\begin{aligned}
&Subterm : (\Lambda s; \Lambda s)Set\\
&\quad Subterm(h, Var(n)) = Empty\\
&\quad Subterm(h, st) = Or(I(h, s), I(h, t))\\
&\quad Subterm(h, \lambda t) = I(Sterm, h, t)\\
&\quad Subterm(h, s\sigma^j t) = Or(I(h, s), I(h, t))\\
&\quad Subterm(h, \varphi^i_k s) = I(h, s)
\end{aligned}$$

Remark 9.2.6 *This induction principle is proved in ALF. To prove it, we first need to prove that:*

For any $a, b \in \Lambda s$. If $a \in SN_s$ and $Subterm(b, a)$, then $b \in SN_s$.

which is proved in the same way as proving lemma 9.2.2.

We could not use the lexicographic induction (see [31]) by formalising the "dpth" in section 3.2, because "dpth" is a partial function and hard to define.

Using this induction principle, we need to solve the following problem:

$$\frac{n : \Lambda s; SN_s(n); ih : (x : \Lambda s; Or(OneStep(n,x), SubTerm(x,n)))P(x)}{? : SN_s(\varphi^i_k n)[i, k : N]}$$

The problem $? : SN_s(\varphi^i_k n)[i, k : N]$ is solved by introduction rule of SN_s and to solve the problem:

$$\frac{\begin{array}{c} n : \Lambda s; SN_s(n); ih : (x : \Lambda s; Or(OneStep(n,x), SubTerm(x,n)))P(x) \\ i, k : N; b : \Lambda s; OneStep(\varphi^i_k n, b) \end{array}}{? : SN_s(b)}$$

This problem is solved by analysing the reduction $OneStep(\varphi^i_k n, b)$, four cases arise:

1. $b = \lambda(\varphi^i_{k+1} a)$ and $n = \lambda(a)$. This is solved by lemma 9.2.2 and I.H. because a is a subterm of n.

2. $b = (\varphi^i_k a_1)(\varphi^i_k a_2)$ and $n = a_1 a_2$. This is proved by lemma 9.2.1 and I.H. because both a_1 and a_2 are subterms of n.

3. $b = Var(j)$ for some $j \in N$. This is obviously true.

4. $b = \varphi^i_k(n')$ and $OneStep(n, n')$. This is solved by I.H.

$\square$

To prove Lemma 9.2.4, we need the following induction principle, which corresponds to induction on $(dpth(n), lgth(n), dpth(m), lgth(m))$:

$$
\begin{aligned}
&LexicoInduc : (P : (Sterm; Sterm)Set\\
&\quad wf\_ih : (n, m : Sterm; SN(n); SN(m);\\
&\qquad ih : (x, y : Sterm;\\
&\qquad\quad ((OneStep(n, x) \vee Subterm(x, n)) \wedge I(y, m))\\
&\qquad\quad \vee (I(x, n) \wedge (OneStep(m, y) \vee Subterm(y, m)))))P(x, y))P(n, m)\\
&\quad n : Sterm; m : Sterm; SN(n); SN(m))P(n, m)
\end{aligned}
$$

9.3 Proving SN of s via SN of σ

We show strong normalisation of s by giving a translation from s to σ in ALF in this section.

Theorem 3.2.10 reads in ALF:

$$SigmaSimulateS : (a, b : \Lambda; p : OneStep(a, b))SgReduceTo(TranStoSgT(a), TranStoSgT(b))$$

where TranStoSgT is the translation function T, $OneStep(a, b)$ means one step reduction $a \rightarrow b$, and $SgReduceTo(a, b)$ means $a \rightarrow^+_\sigma b$.

The ALF proof of the theorem above is by case analysis on the proof object p. We should check when any of the seven *S_OneStep* rules for p, the theorem is correct.

One of the main tasks is, when coming to the rule *RuleDirect*, to prove that theorem holds for any of the reduction rules(*SRules*). We can use induction to prove other cases, i.e. the compatible rules.

For instance, we should prove the following propositions hold when coming to the σ-destruction:

$Projection_1 : (n : N; b : \Lambda\sigma^t)SgReduceTo(n[b_n], b[\uparrow^n])$

$Projection_2 : (n, i : N; i > n; b : \Lambda\sigma^t)SgReduceTo(n[b_i], n)$

$Projection_3 : (n, i : N; n > i; b : \Lambda\sigma^t) SgReduceTo(n[b_i], n-1)$

It is easy to see that they are true from the intuition. However when proving them in ALF, there are a lot of details which we need to do. Let's take these projections to see some details of the ALF proof.

First of all, we should have a denotation for b_i for any $i \geq 1$. When we write b_i, we actually refer to a function $b_i : \Lambda\sigma^t \times N \to \Lambda\sigma^s$. In the following we will feel free to use the convention notation b_i.

This is defined as follows:

$b_i = ConcaFinite(i, b[\uparrow^{i-1}] \cdot \uparrow^{i-1})$

where

$$\begin{aligned} & ConcaFinite : (n : N; s : \Lambda\sigma^s)\Lambda\sigma^s \\ & \quad ConcaFinite(0, s) = s; \\ & \quad ConcaFinite(n+1, s) = ConcaFinite(n, n \cdot s) \end{aligned}$$

Alternatively we would have defined b_i by two simultaneously defined functions LiftSubs and LiftSterm:

$b'_1 = b[id] \cdot id$

$b'_{i+1} = 1 \cdot Liftsubs(b'_i)$

where

$$\begin{aligned} LiftSubs \;\; & : (s : \Lambda\sigma^s)\Lambda\sigma^s \\ & LiftSubs(id) = \uparrow \\ & LiftSubs(\uparrow) = \uparrow \circ \uparrow \\ & LiftSubs(a \cdot s) = LiftSterm(a) \cdot LiftSubs(s) \\ & LiftSubs(s \circ t) = s \circ LiftSubs(t) \end{aligned}$$

$$\begin{aligned} LiftSterm : \;\; & (a : \Lambda\sigma^t)\Lambda\sigma^t \\ & LiftSterm(1) = 1[\uparrow] \\ & LiftSterm(ab) = LiftTerm(a)LiftTerm(b) \\ & LiftSterm(\lambda a) = \lambda(a[1 \cdot (\uparrow \circ \uparrow)]) \\ & LiftTerm(a[s]) = a[LiftSubs(s)] \end{aligned}$$

Lemma 9.3.1 *1. $a[\uparrow] \to_\sigma^\star LiftSterm(a)$ for any term $a \in \Lambda\sigma^t$;*

2. $s \circ \uparrow \to_\sigma^\star LiftSubs(s)$;

3. $LiftSubs(\uparrow^n) = \uparrow^{n+1}$;

4. $LiftSterm(n) = n + 1$;

In fact we can prove that they are two different notations for b_i.

Lemma 9.3.2 *$ConcaFinite(n+1, LiftSubs(s)) = 1 \cdot LiftSubs(ConcaFinite(n,s))$ for any $s \in \Lambda\sigma^s$.*

Lemma 9.3.3 *$b'_{n+1} = ConcaFinite(n, b[\uparrow^n] \cdot \uparrow^n)$.*

The latter notations for b_i seem more natural and immediate, by introducing the two functions, but we need more lemmas about the properties of the substitutions b_i than the first notations. Anyway, in any case it is our intuition which we must ask to convincing us that these are really denoting b_i.

$Projection_1$ is proved by the following lemma:

Lemma 9.3.4 *$\uparrow^n \circ Confinite(n,s) \to_\sigma^+ s$ for any $s \in \Lambda\sigma^s$ and $n > 0$.*

It is easy to prove it by induction on n.

Now we can solve $Projection_1$ by induction on n.

For n = 1, $1[b_1] = 1[b[id] \cdot id] \to b[id] = b[\uparrow^0]$ by the rule VarCons.

For n+1, $(n+1)[b_n] = 1[\uparrow^n][b_n] \to 1[\uparrow^n \circ b_n] \to 1[b[\uparrow^n] \cdot \uparrow^n] \to b[\uparrow^n]$ by the rule Clos and Lemma 9.3.4. This complete the proof of $Projection_1$.

Lemma 9.3.5 *$Projection_1 : (n : N; n > 0; b : \Lambda\sigma) SgReduceTo(n[b_n], b[\uparrow^n])$*

$Projection_2$ is proved by induction on n.

For n = 1 we need to prove: $1[ConcaFinite(b,i)] \to 1$

This is proved by the rule VarCons in one step. This is because $b_i = 1 \cdot s$ for some substitution s when $i > 1$. However, b_i is defined by the function ConcaFinite, we need to prove this fact. It is immediate if we use the notation b'_i.

For n+1, we should prove:

$1[\uparrow^n][b_i] \to_\sigma^+ 1[\uparrow^n]$

By the closure rule, we have:

$1[\uparrow^n][b_i] \to_\sigma 1[\uparrow^n \circ b_i]$

Intuitively $\uparrow^n \circ b_i = (n+1) \cdot \ldots \cdot b[\uparrow^{i-1}] \cdot \uparrow^{i-1}$, and $Projection_2$ is solved by the rule VarCons.

Therefore we should prove that $\uparrow^n \circ b_i = (n+1) \cdot ... \cdot b[\uparrow^{i-1}] \cdot \uparrow^{i-1}$

Now there should be a notation for $(n+1) \cdot ... \cdot b[\uparrow^{i-1}] \cdot \uparrow^{i-1}$

So we define another notation ConcaFinite3:

$$\begin{aligned} ConcaFinite3 \quad & : (n, i : N; s : \Lambda\sigma^s)\Lambda\sigma^s \\ & ConcaFinite3(n, 0, s) = n \cdot s \\ & ConcaFinite3(n, i+1, s) = ConcaFinite3(n, i, (n+i+1) \cdot s) \end{aligned}$$

Lemma 9.3.6 *$ConcaFinite3(0, n, s) = ConcaFinite(n+1, s)$ for $n \in N$ and $s \in \Lambda\sigma^s$.*

Lemma 9.3.7 *$\uparrow^n \circ ConcaFinite(i, s) \rightarrow^+_\sigma ConcaFnite3(n, i-n-1, s)$, for $i > n$, and $s \in \Lambda\sigma^s$.*

We have to prove that $ConcaFinite3(n, i, s)$ has the form $n \cdot s$ for some substitution s explicitly.

Lemma 9.3.8 *$ConcaFinite3(n, i, s) = n \cdot LS1(n, i, s)$, for any $i, n \in N$ and $s \in \Lambda\sigma^s$. where $LS1(n, 0, s) = s$ and $LS1(n, i+1, s) = ConcaFinite3(n+1, i, s)$.*

Lemma 9.3.9 *$Projection_2 : (n, i : N; i > n; b : \Lambda\sigma)SgReduceTo(n[b_i], n)$*

$Projection_3$ is proved by induction on the proof object $p : n > i$ based on the following lemma:

Lemma 9.3.10 *$\uparrow^{n+1+i} \circ b_i \rightarrow^+_\sigma \uparrow^{n+i}$ for any $i, n \in N$.*

This is because $n[b_i] = 1[\uparrow^{n-1}][b_i] \rightarrow 1[\uparrow^{n-1} \circ b_i] \rightarrow 1[\uparrow^{n-2}]$

Lemma 9.3.11 *$Projection_3 : (n, i : N; n > i; b : \Lambda\sigma)SgReduceTo(n[b_i], n-1)$*

Having proved that T is a strict interpretation, we get:

Theorem 9.3.12 *calculus s is strongly normalising.*

Chapter 10

Conclusion and Future Work

10.1 Conclusion

We give formal proofs of strong normalisation of σ and s in ALF. To prove σ is strongly normalising, we have formalised all the notions and checked all the proofs in [11], some of which were quite informal and needed to be checked formally, e.g. 7.3.6. For the calculus s, we gave three formal proofs of strong normalisation, which are the usual ways to prove strong normalisation of explicit substitutions. Two of the formalisations were presented in this thesis.

In this thesis we have tried to investigate the process of formalising a proof, which is already written on paper, to a formal proof in a proof checker. We have tried to formalise as much as possible the processes of proving strong normalisations of substitutions, and to illustrate there are some subtleties need to be clarified and more work is expected.

A lot of work has been done on proof checking in various proof checkers, such as ALF, Coq, Lego and etc. There are several reasons for doing so. First of all, we are interested in if our proofs of theorems are really correct, especially for those proofs which are difficult and intricate. Theorem checkers are very strict, they do not take anything granted which usually lead people to a wrong proof. The termination proof of σ by induction is very difficult and intricate. The formalisation of the termination proof of σ gives us assurance that the substitutions in σ terminate.

The second reason is to help people prove theorems whose proofs are cumbersome by hands. In the proof process, one only give some orders to the prover and the prover carries out the detailed computations and reasonings. For instance, when filling a hole, the user can give only the name of the lemma and the prover will fills in all the parameters itself by pattern matching. In the case of proving termination of explicit substitutions where there are many reduction rules and one need to carry out many reductions in the proof, ALF is really helpful. Once getting used to the system and having introduced the definitions, I now find it more cumbersome to do the proofs by hand. However, one need to do more work to give a formal proof. In the strong normalisation proof of σ, there are many lemmas which are true intuitively, but involve much more work to prove them formally. For instance, in Lemma 7.3.6, the position of a redex can occur in three cases; in Lemma 7.2.19, when replacing a λ-hole in a context C, which is very good for s, with a L-term t, the result context is very good for the term replacing the sub-term at the same position with t.

Another reason is to investigate the processes of the mathematical proofs, help people understand the mathematical reasoning and build automatic theorem provers. In fact, it was during the implementation of theorem provers and proof checkers and checking proofs that one understands more about mathematical proofs. In the case of termination of explicit substitutions, we hope to understand why for some calculi decreasing measures can be found and for others they can not be found.

10.2 Future work

As we have not found an ideal explicit substitution calculus and a new one is always coming into being, it will be interesting to find out a general way to prove properties of explicit substitutions such as strong normalisation, confluence and preserving β-strong normalisation and further to develop a package of special tools to deal with calculi with explicit substitutions, e.g. to help researchers to prove the above properties. We hope that our work will help to understand the termination process and find a general way to prove termination of calculi of explicit substitutions. We believe that our proof can be adapted to check termination of other existing calculi of substitutions. One interesting work to do is to check PSN proofs, confluence proofs of $\lambda\sigma$, λs and other calculi.

Bibliography

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs.* The MIT Press, Cambridge, Mass., second edition, 1996.

[3] T. Altenkirch. A formalisation of the strong normalisation proof for system F in LEGO. In *Lecture Notes in Computer Science*, volume 664, pages 13–28. Springer-Verlag, 1993.

[4] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics, Revised edition.* North Holland, 1984.

[5] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, September 1996.

[6] M. Bezem, J. W. Klop, and V. van Oostrom. Diagram techniques for confluence. *Information and Computation.* Accepted for publication in *Information and Computation.* Final manuscript received for publication August 22, 1997.

[7] C. Coquand. From semantics to rules: A machine assisted analysis. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proceedings of the 7th Workshop on Computer Science Logic*, pages 91–105. Springer-Verlag LNCS 832, 1993.

[8] T. Coquand. Pattern matching with dependent types. In B. Nordström, K. Pettersson, and G. Plotkin, editors, *Informal Proceedings Workshop on Types for*

Proofs and Programs, Båstad, Sweden, 8–12 June 1992, pages 71–84. Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ., 1992.

[9] T. Coquand, B. Nordström, J. M. Smith, and B. von Sydow. Type theory and programming. *Bulletin of the European Association for Theoretical Computer Science*, 52:203–228, February 1994. Columns: Logic in Computer Science.

[10] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 2nd edition, 1993. (1st ed., Pitman Publishing, London, and J. Wiley and Sons, New York).

[11] P-L Curien, T.Hardin, and A.Rı́os. Strong normalisation of substitutions. *Logic and Computation*, 6:799–817, 1996.

[12] N. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Technical report, Department of Mathematics , University of Eindhoven, Netherlands, 1978.

[13] N. Dershowitz. Termination of rewriting. In J.-P. Jouannaud, editor, *Rewriting Techniques and Applications*, pages 69–115. Academic Press, 1987. Reprinted from *Journal of Symbolic Computation.*

[14] P. Dybjer. Inductive families. *Formal Aspects of Computing*, 6(4):440–465, 1994.

[15] T. Hardin and A. Laville. Proof of termination of the rewriting system SUBST on CCL. *Theoretical Computer Science*, 46(2-3):305–312, 1986.

[16] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. *France-Japan Artificial Intelligence and Computer Science Symposium*, December 1989.

[17] S. L. Peyton Jones. *The Implementation of Functional Programming Languages.* Computer Science. Prentice-Hall, 1987.

[18] F. Kamareddine and R. P. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, pages 197–240, 1993.

[19] F. Kamareddine and A. Ríos. Weak normalisation of simply typed λs_e on open terms. *Reports in the Department of Computing Science.*

[20] F. Kamareddine and A. Ríos. A λ-calculus a la de bruijn with explicit substitutions. In *PLILP95, Lecture Notes in Computer Science*, volume 982, pages 45–62. Springer-Verlag, 1995.

[21] F. Kamareddine and A. Ríos. Bridging the $\lambda\sigma$- and λs-styles of explicit substitutions. Technical report, Department of Computing Science, Glasgow University, 1997.

[22] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, July 1997.

[23] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$: a journey through calculi of explicit substitutions. In ACM, editor, *Proceedings of 21st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 60–69, New York, NY, USA, January 1994. ACM Press.

[24] L. Magnusson. *The Implementation of ALF—A Proof Editor Based on Martin-Löf's Monomorphic Type Theory with Explicit Substitution.* PhD thesis, Chalmers University of Technology and Göteborg University, January 1995.

[25] P. Martin-Löf. An intuitionistic theory of types. *Logic Colloquium '73*, 1975.

[26] J. McKinna and R. Pollack. Pure type systems formalised. In M. Bezem and J. F. Groote, editors, *Proceedings 1st Intl. Conf. on Typed Lambda Calculi and Applications, TLCA'93, Utrecht, The Netherlands, 16–18 March 1993*, volume 664 of *Lecture Notes in Computer Science*, pages 289–305. Springer-Verlag, Berlin, 1993.

[27] P. A. Mellies. Typed lambda-calculi with explicit substitutions may not terminate. In *Lecture Notes in Computer Science*, volume 902, pages 328–334. Springer-Verlag, 1995.

[28] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus (extended abstract). In *Proceedings of the Eleven Annual IEEE Symposium on Logic in Computer Science*, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[29] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's type theory : An Introduction.* Oxford: Clarendon, 1990.

[30] L. C. Paulson. Isabelle: The next 700 theorem provers. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[31] R. Pollack. The theory of lego: A proof checker for the extended calculus of constructions. 1994. PhD thesis.

[32] A. Ríos. *Contribution à l't́ude des λ-calculs avec substitutions explicites.* PhD thesis, Université de Paris 7, 1993.

[33] A. Saïbi. Formalisation of a λ-calculus with explicit substitutions in Coq. In P. Dybjer, B. Nordström, and J. Smith, editors, *Proceedings of the International Workshop on Types for Proofs and Programs*, pages 183–202, Båstad, Sweden, June 1994. Springer-Verlag LNCS 996.

[34] A. Tasistro. *Substitution, record types and subtyping in type theory, with applications to the theory of programming.* Phd thesis, Dept. of Computing Science, Chalmers Univ. of Technology and Univ. of Göteborg, 1997.

[35] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, January 1994. Conditional term rewriting systems (Pont-à-Mousson, 1992).