Classical logic, control calculi and data types

Robbert Krebbers[1]
Radboud University Nijmegen

Master's Thesis
in Foundations of Computer Science

Date:
October, 2010

Supervised by:
prof. dr. Herman Geuvers (herman@cs.ru.nl)
dr. James McKinna (james@cs.ru.nl)

Thesis number:
643

[1] mail@robbertkrebbers.nl

Summary

This thesis is concerned with the relation between classical logic and computational systems. For constructive logic we have the well-known Curry-Howard correspondence, which states that there is a correspondence between formulas and types, proofs and programs, and proof normalization and reduction. But until quite recently, people believed that this correspondence was limited to constructive logic.

Nonetheless already quite old results by Kreisel, Friedman and Gödel show that certain classical proofs do contain computational content. However, these results use a translation of classical proofs into constructive logic and do not describe a direct correspondence between classical logic and computation. A direct correspondence remained unknown until 1990, when Griffin extended the Curry-Howard correspondence to classical logic by incorporation of Felleisen's control operator $\mathcal{C}$.

In this thesis we continue on Griffin's track. In the first part we investigate various control calculi: Felleisen's $\lambda_{\mathcal{C}}$-calculus with Griffin's typing rules, Rehof and Sørensen's λ_{Δ}-calculus and Parigot's λ_{μ}-calculus. We are especially interested in the main meta-theoretical properties: confluence, normal form theorems, subject reduction and strong normalization. Our research will indicate that both the $\lambda_{\mathcal{C}}$-calculus and λ_{Δ}-calculus suffer from various defects.

Since none of the discussed systems contain data types, we will extend Parigot's λ_{μ}-calculus with a data type for the natural numbers and a construct for primitive recursion in the second part of this thesis. We prove that our system satisfies subject reduction, has a normal form theorem, is confluent and strongly normalizing. The last two proofs require various niceties to make the standard proof methods work.

The long term goal of the research initiated in this thesis is to develop a system that supports a limited amount of classical reasoning and also contains dependent and inductive types. Such a system would have two major applications.

1. It could be used to prove the correctness of programs with control,
2. It could be used to obtain programs with control by program extraction from classical proofs.

Contents

Chapter 1

Introduction

The *Curry-Howard correspondence* states that there is a correspondence between logic and computational systems: formulas correspond to types, proofs to programs and proof normalization to reduction. This is an amazing result, because it relates logic, which is generally considered as static, to computation, which is generally considered as dynamic.

For quite some time this correspondence has been employed for the development of provably correct functional programs. In particular, it has led to the development of dependently typed λ-calculi. We can use these systems in two ways so as to construct correct programs.

Correctness proofs. The system is used as a functional programming language to write a certain program. Meanwhile, its rich type structure is used to state the program's specification and then one proves that the program is correct with respect to its specification.

Program extraction. The system is used to state a specification and then one proves that a solution to that specification exists. Now, by removing computationally irrelevant parts, a program that is guaranteed to be correct with respect to its specification is extracted.

An example of such a formal system is the *Calculus of Constructions (CC)* and its extension the *Calculus of Inductive Constructions (CIC)*, which adds support for inductive types. This system is the basis of the interactive proof assistant Coq [CDT].

The Curry-Howard correspondence is originally considered with respect to constructive logic and not with respect to classical logic. In fact, until quite recently, people believed that the Curry-Howard correspondence was limited to constructive logic. Nonetheless already quite old results by Kreisel, Friedman and Gödel [Kre58, Fri78] show that certain classical proofs do contain computational content. More precisely, they have shown that provability of a Π^0_2-formula in Peano arithmetic and Heyting arithmetic coincides. However, these results use a translation of classical proofs into constructive logic and do not describe a direct correspondence between classical logic and computation.

A direct correspondence remained unknown until 1990, when Griffin [Gri90] extended the Curry-Howard correspondence to classical logic by incorporation of

control operators. An interesting part of Griffin's work is that CPS-translations, which allow simulation of control operators in a system without it, correspond to logical embeddings of classical logic into constructive logic.

Unfortunately, the correspondence for classical logic is generally studied with respect to quite simple systems. Most of the systems present in today's literature do not contain inductive types, or actually, most do not even contain data types for basic types as the natural numbers.

The long term goal of the research initiated in this thesis is to develop a system that supports a limited amount of classical reasoning and also contains dependent and inductive types. Such a system would have two major applications.

1. It could be used to prove the correctness of programs with control,
2. It could be used to obtain programs with control by program extraction from classical proofs.

In this thesis we make a start by extending a system with control, namely Parigot's λ_μ-calculus [Par92], with data types.

1.1 Control mechanisms

Control mechanisms, also known as exception mechanisms, allow to clearly separate the unusual case from the normal case and can moreover help to write more efficient programs. We distinguish the following kind of unusual cases [Goo75].

1. Range failures. These failures occur when an operation is unable to satisfy its postcondition. An example of such a failure is an IO-error while reading or writing to a file.
2. Domain failures. These failures occur when an operation has been given input that does not match its precondition. For example, such failure might occur when one attempts to take the tail of an empty list or tries to divide by zero.

In this thesis we solely consider domain failures because we study formal systems instead of actual programming languages. Range failures cannot occur in the considered formal systems because external dependencies as IO do not appear. Furthermore, domain failures can help to write more efficient programs. We will illustrate this by considering the following simple functional program[1].

```
let rec listmult l = match l with
  | nil    -> 1
  | x :: k -> x * (listmult k)
```

[1] Control mechanisms will most likely not improve the performance of this particular program considered in a general purpose programming language, because arithmetic is performed directly by the machine and thus very efficient. However, for more complex programs or data types, control mechanisms will certainly help to improve the performance. For example, in [CGU00] it is used to incorporate a sophisticated backtracking technique. But so as to keep it simple we restrict ourselves to toy examples.

This program takes a list of natural numbers and yields the product of its elements. If a list contains an element whose value is zero, this function yields zero. Unfortunately, when a zero is encountered, it is multiplied by all elements in the list. One could try to optimize this function by letting it stop multiplying once a zero is encountered.

```
let rec listmult l = match l with
  | nil    -> 1
  | 0 :: k -> 0
  | x :: k -> x * (listmult k)
```

However, on its way out of the recursion, a zero is still multiplied by all elements that were previously encountered. Instead, it would be nice if we could break out of the recursion once a zero is encountered. We will incorporate Lisp's control operators `catch` and `throw` to achieve this goal. First we describe the intuitive semantics of these operators.

Evaluation of the term `catch` α t results in evaluation of t. If evaluation of t yields an actual result v, then `catch` α t yields v. In this case, we say that t *returns normally*. However, if we encounter a term `throw` α s during the evaluation of t, then `catch` α t yields s. In this case, we say that t *returns exceptionally*.

Now, by incorporation of the control operators `catch` and `throw`, we let our program break out of the recursion once a zero is encountered.

```
let listmult l = catch α (listmult2 l)

let rec listmult2 l = match l with
  | nil    -> 1
  | 0 :: k -> throw α 0
  | x :: k -> x * (listmult2 k)
```

Here, the function `listmult2` is not defined for lists that contain an element whose value is 0. So, if supplied with an list that contains an element whose value is 0 one can say that a domain failure occurs.

In order to reason formally about functional programs one could cast them into a formal framework, for example the λ-calculus. The ordinary λ-calculus does, however, not support control mechanisms. Fortunately, initiated by the work of Felleisen *et al.* [FF86, FFKD87], various extensions of the λ-calculus with control mechanisms have been developed.

More surprisingly, Griffin [Gri90] discovered that control mechanisms can be typed with classical proof rules and thereby extend the Curry-Howard correspondence to classical logic.

1.2 Our approach

In the first part of this thesis (Chapter 3) we investigate various control calculi: Felleisen's $\lambda_{\mathcal{C}}$-calculus [FF86, FFKD87] with Griffin's typing rules [Gri90], Rehof and Sørensen's λ_{Δ}-calculus [RS94] and Parigot's λ_{μ}-calculus [Par92]. By studying these systems we determine which one is suitable for an extension with

data types. However, by no means we claim that our investigation is exhaustive, because there are simply too many control calculi present in today's literature. Hence it is definitely possible that a more suitable system, which has not been considered in this thesis, exists.

For each system we take a look at the desired meta theoretical properties. We are especially interested in confluence, normal form theorems, subject reduction and strong normalization. Based on our investigations we will indicate some major defects in both the $\lambda_{\mathcal{C}}$- and λ_{Δ}-calculus. Furthermore, we will show that the λ_{μ}-calculus satisfies the main theoretical properties and is able to simulate `catch` and `throw`.

In the second part (Chapter 4) we present our main technical contribution: a Gödel's $\mathbf{T}$ version of the λ_{μ}-calculus, which we name $\lambda_{\mu}^{\mathbf{T}}$. Gödel's $\mathbf{T}$ is simple type theory extended with a base type for the natural numbers and a construct for primitive recursion. The $\lambda_{\mu}^{\mathbf{T}}$-calculus is, however, not a straightforward "merge" of λ_{μ} and Gödel's $\mathbf{T}$. Some of its reduction rules closer are closer to a call-by-value system than to the ones one would expect of a call-by-name system (which λ_{μ} originally is). Firstly, it contains the reduction rule $\mathtt{S}\mu\alpha.c \to \mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)]$, which is necessary in order to maintain a normal form theorem. Secondly, in order to unfold $\mathsf{nrec}\ r\ s\ (\mathtt{S}t)$ we have to reduce t to an actual numeral. This is required because it would otherwise result in a loss of confluence.

In the second part we moreover prove that $\lambda_{\mu}^{\mathbf{T}}$ satisfies subject reduction, has a normal form theorem, is confluent and strongly normalizing. The last two proofs are quite non-trivial because various niceties are required to make the standard proof methods work.

Our confluence proof uses the notion of parallel reduction and defines the complete development of each term. Surprisingly, the author was unable to find a confluence proof for the original untyped λ_{μ}-calculus. In [BHF01] there is a confluence proof for λ_{μ} without the $\to_{\mu\eta}$-rule ($\mu\alpha.[\alpha]t \to_{\mu\eta} t$ provided that $\alpha \notin \mathrm{FCV}(t)$). Although [BHF01] suggests how to extend parallel reduction for the $\to_{\mu\eta}$-rule, a definition of the complete development and a proof are absent. In this thesis we extend the methodology of [BHF01] to the case of $\lambda_{\mu}^{\mathbf{T}}$, which also includes the $\to_{\mu\eta}$-rule.

Our strong normalization proof proceeds by defining reductions $\to_A$ and $\to_B$ such that $\to = \to_A \cup \to_B$. First we prove that $\to_A$ is strongly normalizing by the reducibility method. Secondly, we prove that $\to_B$ is strongly normalizing and moreover that each infinite $\to_{AB}$-reduction sequence can be transformed into an infinite $\to_A$-reduction sequence. The first phase is inspired by Parigot's proof of strong normalization for λ_{μ} [Par97] and the second phase is inspired by Rehof and Sørensen's proof of strong normalization for λ_{Δ} [RS94].

Moreover we show that our system can be embedded into Gödel's $\mathbf{T}$ and into the $\lambda_{\mu}^{\mathbf{2}}$-calculus, which is a polymorphic variant of Parigot's λ_{μ}-calculus [Par92]. Finally we present a program in our system and prove that it is correct with respect to its specification.

1.3 Related work

To the author's knowledge there is little evidence of research in which control calculi are combined with data types. Also, there is little evidence of research in which program extraction, by removing computational irrelevant parts, from classical proofs is discussed. We will summarize relevant research.

Murthy considered a system with control operators, arithmetic, products and sums in his PhD thesis [Mur90]. However, he was mainly concerned with CPS-translations while we would like to reason directly about programs in our system.

Parigot has described a second-order variant of his λ_μ-calculus [Par92]. Although this system is very powerful, it suffers from the same weakness as System **F**, namely its efficiency is pretty poor. Also, as noticed in [Par92, Par93], this system does not ensure unique representation of data types. For example, there is not a one-to-one correspondence between natural numbers and closed normal forms of the type of Church numerals.

Berger, Buchholz and Schwichtenberg have described a form of program extraction from classical proofs [BBS00]. Their method extracts a Gödel's **T** term from a classical proof in which all computational irrelevant parts are removed. To prove the correctness of their approach they have given a realizability interpretation. However, since their target language is Gödel's **T**, resulting programs do not contain control mechanisms.

Caldwell, Gent and Underwood considered program extraction from classical proofs in the proof assistant NuPrl [CGU00]. In their work they extended NuPrl with a proof rule for Peirce's law and associated `call/cc` for its extraction. Now, program extraction indeed results in a program with control. However, the authors were mainly interested in using program extraction to obtain efficient search algorithms and did not prove any meta theoretical results so it is unclear whether their approach is correct for arbitrary classical proofs. In fact, in Appendix A we will repeat their approach in the proof assistant Coq and show that we can construct incorrect programs.

Herbelin developed an intuitionistic predicate logic that supports a limited amount of classical reasoning [Her10]. His system proves a variant of Markov's principle, which is computationally associated with an exception mechanism. Although his system does not contain data types and realizability is left for further research, it seems like a good candidate to extend with data types. As this research is very recent (July 2010), we have not been able to incorporate it into our work, which was by then nearly finished.

1.4 Outline

We give a brief overview of the contents of each chapter.

- Chapter 2 discusses the background that is required to read this thesis. On the one hand we present some logics and on the other hand some typed λ-calculi. We will relate these seemingly unrelated notions by means of the Curry-Howard correspondence, which establishes a correspondence between terms and proof and between types and logical formulas.

Furthermore, this chapter presents some important meta theoretical properties of the systems: confluence, normal form theorems, subject reduction and strong normalization. These properties will return when we discuss various other systems in the subsequent chapters.

- Chapter 3 considers extensions of the λ-calculus with control operators. Control operators allow to create more efficient programs and extend the Curry-Howard correspondence to classical logic. We will present various systems and show that for some of them important meta theoretical properties, which we have discussed in Chapter 2, fail. Also, this chapter discusses CPS-translations, which allow simulation of control operators in a system without it.

- Chapter 4 deals with extensions of the λ-calculus with both basic data types and control operators. This is a fairly new topic in the literature, because such extensions are usually discussed separately. This chapter includes the main result of this thesis: an extension of Gödel's **T** with control operators. We will prove various non-trivial meta theoretical properties of this system. Also, we will show how this system relates to previously discussed systems by means of translations into those systems. Finally, we will consider an example program and prove its correctness.

- Chapter 5 discusses some extensions of our work. First we discuss extensions with other data types, as lists and products. Secondly, we introduce program extraction and indicate how our system could be used for extraction of programs with control from classical proofs.

- Chapter 6 finishes with conclusions.

Acknowledgements

I would like to thank Herman Geuvers and James McKinna for their supervision, without their help this thesis would not be as it is now. I would also like to thank Herman Geuvers for initiating this project by introducing me to the Curry-Howard correspondence for classical logic. Moreover, I would like to thank Hugo Herbelin for answering various questions.

Chapter 2

Background

In this chapter we will briefly introduce various well-known notions that play an important role in this thesis. In Section 2.1 we discuss minimal, intuitionistic and classical variants of first-order propositional logic, in Section 2.2 we discuss the untyped λ-calculus, in Section 2.3 we discuss the simply-typed λ-calculus and the Curry-Howard correspondence with first-order minimal logic, in Section 2.4 we discuss minimal and classical variants of second-order propositional logic and in Section 2.5 we discuss the second-order typed λ-calculus.

2.1 First-order propositional logic

In this section we present minimal, intuitionistic and classical first-order propositional logic and the relation between these logics.

Definition 2.1.1. Minimal first-order propositional formulas *are built from an infinite set of atoms* $(X, Y, \ldots)$ *and an implication arrow* $(\rightarrow)$.

$$A, B ::= X \mid A \rightarrow B$$

Moreover, an environment $(\Gamma, \Sigma, \ldots)$ *is a finite set of formulas.*

Definition 2.1.2. *A judgment* $\Gamma \vdash A$ *is derivable in* minimal first-order propositional logic *if it can be derived by the natural deduction rules shown in Figure 2.1.*

$$\frac{A \in \Gamma}{\Gamma \vdash A} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \qquad \frac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

(a) axiom (b) $\rightarrow_i$ (c) $\rightarrow_e$

Figure 2.1: Natural deduction for minimal first-order propositional logic.

By saying that $\Gamma \vdash A$ is derivable we mean that there exists a finite tree whose nodes are labeled by judgments $\Sigma \vdash B$ such that [SU06]:

1. The root is labeled by $\Gamma \vdash A$,

2. Each leaf is labeled by an axioms of the system,
3. Each label of a parent node is obtained from the labels of its children using one of the rules of the system.

For example, let $\Gamma = \{A \to B \to C, A \to B\}$, now the following tree is a valid derivation.

$$\dfrac{\dfrac{\dfrac{\Gamma, A \vdash A \to B \to C \qquad \Gamma, A \vdash A}{\Gamma, A \vdash B \to C} \qquad \dfrac{\Gamma, A \vdash A \to B \qquad \Gamma, A \vdash A}{\Gamma, A \vdash B}}{\Gamma, A \ \vdash \ C}}{\Gamma \ \vdash \ A \to C}$$

However, if we let $\Gamma = \{(A \to B) \to A\}$ and $A \neq B$, then the following tree is not a valid derivation because $\Gamma, A \vdash B$ is not an axiom.

$$\dfrac{\Gamma \ \vdash \ (A \to B) \to A \qquad \dfrac{\Gamma, A \ \vdash \ B}{\Gamma \ \vdash \ A \to B}}{\Gamma \ \vdash \ A}$$

In fact, there does not even exist a derivation of the judgment $(A \to B) \to A \vdash A$ in minimal classical logic as we will obtain by Theorem 2.1.7.

As we have seen, minimal first-order propositional formulas do merely consist of atoms and arrows. Now we present *intuitionistic logic*, in which we extend formulas with a nullary connective false ($\bot$).

Definition 2.1.3. First-order propositional formulas *are minimal propositional formulas extended with a unary connective false ($\bot$).*

$$A, B ::= X \mid \bot \mid A \to B$$

In minimal logic one could treat the connective false ($\bot$) as any other atom. Obviously, since there is no additional rule for that atom, it does not have an exceptional meaning. However, in intuitionistic logic we will add an additional rule. The connective false should be interpreted as *absurd*, that means, any formula can be derived from it.

Definition 2.1.4. *A judgment holds in* intuitionistic first-order propositional logic *if it can be derived by the rules shown in Figure 2.1 and the* Ex Falso Quodlibet (EFQ) *rule, which is shown in Figure 2.2.*

$$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

(a) Ex Falso Quodlibet

Figure 2.2: Intuitionistic rules.

Notation 2.1.5. *We abbreviate $\neg A$ (*not *A) as $A \to \bot$.*

Definition 2.1.6. *A judgment holds in* minimal classical first-order propositional logic *if it can be derived by the rules shown in Figure 2.1 and* Peirce's law (PL). *A judgment holds in* classical logic *if it can be derived by the rules shown in Figure 2.1 and* double negation (DN). *These rules are shown in Figure 2.3.*

$$\frac{\Gamma \vdash (A \to B) \to A}{\Gamma \vdash A}$$
(a) Peirce's law

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$
(b) Double negation

Figure 2.3: Classical rules.

Theorem 2.1.7. *The following diagram indicates the relations between minimal, intuitionistic and classical first-order propositional logic. An arrow from logic L to K indicates that K is strictly stronger than L.*

$$\begin{array}{ccc} \textit{Minimal logic} & \longrightarrow & \textit{Intuitionistic logic} \\ \downarrow & & \downarrow \\ \textit{Minimal classical logic} & \longrightarrow & \textit{Classical logic} \end{array}$$

Moreover, a judgment $\Gamma \vdash B$ holds in minimal classical logic with the EFQ rule assumed iff $\Gamma \vdash B$ holds in classical logic.

Proof. This is proven in [AH03]. □

2.2 The untyped λ-calculus

In this section we describe the λ-calculus: a formal system for function definition. The λ-calculus is of great interest because in the forthcoming sections we will show that terms in typed λ-calculi correspond to derivations in various logics. Therefore a correspondence between proofs and programs is created.

Definition 2.2.1. Untyped λ-terms *are inductively defined over an infinite set of variables $(x, y, \ldots)$ as follows.*

$$t, r, s ::= x \mid \lambda x.t \mid ts$$

Throughout this thesis we let $\mathrm{FV}(t)$ denote the set of free variables of a term t, let $\mathrm{BV}(t)$ denote the set of bound variables, and let $t[x := r]$ denote capture avoiding substitution of r for x in t. Moreover a term t is *closed* if $\mathrm{FV}(t) = \emptyset$ and *open* if $\mathrm{FV}(t) \neq \emptyset$.

Convention 2.2.2. *In this thesis we will present many different languages, of which many contain binders (like $\lambda x.t$ or $\forall X.A$). However, in such languages, we always consider equality up to α-*equivalence*. That means, expressions that only differ in their bound variables are considered equal. Since we consider expressions up to α-equivalence, we use, when necessary, the* Barendregt convention*. That is, given an expression, we may assume that:*

1. *Bound variables are distinct from free variables,*
2. *Binders always bind fresh variables.*

Definition 2.2.3. *Given a binary relation over λ-terms R, then a binary relation over λ-terms $\hat{R}$ is defined as follows.*

1. *If* $t_1\ R\ t_2$*, then* $t_1\ \hat{R}\ t_2$*.*
2. *If* $t_1\ \hat{R}\ t_2$*, then* $(t_1 s)\ \hat{R}\ (t_2 s)$*.*
3. *If* $s_1\ \hat{R}\ s_2$*, then* $(t s_1)\ \hat{R}\ (t s_2)$*.*
4. *If* $t_1\ \hat{R}\ t_2$*, then* $(\lambda x.t_1)\ \hat{R}\ (\lambda x.t_2)$*.*

We say that $\hat{R}$ *is the* compatible closure *of* R*.*

Definition 2.2.4. *Reduction* $t \to t'$ *on* λ*-terms* t *and* t' *is defined as the compatible closure of the following rule.*

$$(\lambda x.t)r \quad \to_\beta \quad t[x := r]$$

We write $\twoheadrightarrow$ *for the reflexive/transitive closure,* $\twoheadrightarrow^+$ *for the transitive closure and* $=$ *for the reflexive/symmetric/transitive closure. A term is in (*β*-)* normal form *if no (*β*-) reduction steps are possible.*

It is allowed to perform β-reduction steps in arbitrary order at arbitrary places, hence the previous definition does not specify a deterministic reduction strategy. Fortunately, the following theorem has as a consequence that no matter what reduction steps are performed, it is always possible to obtain a unique normal form (if such normal form exists).

Theorem 2.2.5. (Church-Rosser) *β-reduction is confluent, that is, if* $t_1 \twoheadrightarrow t_2$ *and* $t_1 \twoheadrightarrow t_3$*, then there exists a term* t_4 *such that* $t_2 \twoheadrightarrow t_4$ *and* $t_3 \twoheadrightarrow t_4$*.*

Proof. Proven in [Tak95, for example]. □

Theorem 2.2.6. *For arbitrary terms* t_1 *and* t_2 *it is undecidable if* $t_1 = t_2$*.*

Proof. Proven in [SU06, for example]. □

In order to write a compiler for a functional programming language based on the λ-calculus it is desired to specify a deterministic evaluation strategy. Therefore we will make a distinction between *reduction* and *evaluation*. Contrary to reduction, evaluation works on whole terms and should be deterministic. The first evaluation strategy that we describe is *call-by-name* evaluation, this strategy evaluates the leftmost β-redex first. To describe this strategy we define the notion of evaluation contexts.

Definition 2.2.7. *A* call-by-name evaluation context *is defined as follows.*

$$E ::= \Box \mid Et$$

Definition 2.2.8. *Given a call-by-name evaluation context* E *and a term* s*, then* $E[s]$*,* substitution of s for the hole in E*, is defined as follows.*

$$\Box[s] := s$$
$$(Et)[s] := E[s]t$$

Definition 2.2.9. Call-by-name evaluation $t \rhd_\beta t'$ *on* λ*-terms* t *and* t' *is defined as follows.*

$$E[(\lambda x.t)r] \quad \rhd_\beta \quad E[t[x := r]]$$

Here, E *is a call-by-name evaluation context.*

We use the symbol $\triangleright$ instead of $\rightarrow$ in order to clearly distinguish reduction and evaluation. Since we are trying to specify a deterministic evaluation strategy it is important that the evaluation rule $\triangleright_\beta$ can be applied at exactly one place. This is a direct consequence of the following definition and lemma.

Definition 2.2.10. Values *are defined as follows.*

$$v, w ::= x \mid \lambda x.r$$

Lemma 2.2.11. *Each closed λ-term is either a value or can be written uniquely as $E[(\lambda x.t)r]$ where E is a call-by-name evaluation context.*

Another well-known evaluation strategy is *call-by-value* evaluation. This evaluation strategy is used in many functional programming languages, for example Scheme and OCaml. In order to describe this strategy we introduce the call-by-value λ-calculus (henceforth λ_v-calculus) by Plotkin [Plo75]. The λ_v-calculus simply restricts the β-reduction rule in such a way that an argument is evaluated first.

$$(\lambda x.t)v \quad \rightarrow_{\beta_v} \quad t[x := v]$$

Just like the ordinary λ-calculus, the λ_v-calculus is confluent [Plo75].

In order to define an evaluation strategy we define call-by-value evaluation contexts first.

Definition 2.2.12. *A* call-by-value evaluation context *is defined as follows.*

$$E ::= \Box \mid vE \mid Et$$

The notion of substitution of s for the hole in E is extended to call-by-value evaluation contexts in the obvious way.

Definition 2.2.13. Call-by-value evaluation $t \triangleright_\beta t'$ *on λ-terms t and t' is defined as follows.*

$$E[(\lambda x.t)v] \quad \triangleright_{\beta_v} \quad E[t[x := v]]$$

Here, E is a call-by-value evaluation context.

Again we have to verify that this is indeed a deterministic specification.

Lemma 2.2.14. *Each closed λ-term is either a value or can be written uniquely as $E[(\lambda x.t)v]$ where E is a call-by-value evaluation context.*

Call-by-name evaluation is usually less efficient than call-by-value evaluation because an abstraction could duplicate its argument. As a result, arguments may be evaluated multiple times. However, as a consequence of the the following lemma this strategy has also some advantages.

Lemma 2.2.15. *If it is possible to reduce a λ-term to a value, then call-by-name evaluation terminates.*

For example, let us consider the term $(\lambda y.\mathsf{I})\,\Omega$ where $\Omega := (\lambda x.xx)\lambda x.xx$. If we perform call-by-value evaluation, then it results in an infinite loop.

$$\begin{aligned}(\lambda y.\mathsf{I})\Omega &\equiv (\lambda y.\mathsf{I})\,((\lambda x.xx)\lambda x.xx)\\ &\triangleright (\lambda y.\mathsf{I})\,((\lambda x.xx)\lambda x.xx)\\ &\triangleright \cdots\end{aligned}$$

However, call-by-name evaluation results in a normal form, because the non-terminating argument Ω is thrown away and therefore not evaluated.

$$(\lambda y.\mathsf{I})\,\Omega \rhd \mathsf{I}$$

Because unused arguments are not evaluated there is great interest in optimizations of call-by-name. *Lazy evaluation* is such optimization, it keeps track of the arguments and evaluates them only the first time they are required. Lazy evaluation is used in functional programming languages as Haskell and Clean. For a nice account of a formal description of a lazy evaluation semantics for the λ-calculus we refer to [Lau93].

2.3 The simply typed λ-calculus

If one considers an untyped λ-term as a function it does not have a fixed domain and range. To resolve this problem, typed versions of the λ-calculus were introduced. These typed λ-calculi form a foundation for many functional programming languages. In this section we describe the simply typed λ-calculus (henceforth $\lambda{\to}$).

Definition 2.3.1. Simple types *are built from an infinite set of type variables* $(\alpha, \beta, \ldots)$ *and an implication arrow* $(\to)$*.*

$$\rho, \delta ::= \alpha \mid \rho \to \delta$$

An environment *(*$\Gamma, \Sigma, \ldots$*) is an association list of types indexed by* λ*-variables.*

The most commonly used presentations of a typed λ-calculus are Church-style and Curry-style [Bar92, Geu08]. In Church-style, each abstraction is annotated by a type, for example $\lambda x : \alpha.\lambda y : \alpha \to \beta.yx$. Here, each term has a unique type because of the type annotations. Systems in Curry-style just use untyped λ-terms, hence abstractions are not annotated by types, for example $\lambda xy.yx$. Terms in such a system can have multiple types, for example $\lambda xy.yx$ can be typed with $\alpha \to (\alpha \to \beta) \to \beta$ and $\alpha \to (\alpha \to \gamma \to \gamma) \to \gamma \to \gamma$. In this thesis we study systems à la Church.

Convention 2.3.2. *Although we study systems à la Church, we omit type annotations when they are obvious or just irrelevant.*

Definition 2.3.3. *The* terms *of* $\lambda{\to}$ *are inductively defined over an infinite set of variables* $(x, y, \ldots)$ *as follows.*

$$t, r, s ::= x \mid \lambda x : \rho.t \mid ts$$

Definition 2.3.4. *A* $\lambda{\to}$-typing judgment $\Gamma \vdash t : \rho$ *denotes that a term* t *has type* ρ *in an environment* Γ*. The derivation rules for such judgments are shown in Figure 2.4.*

As the preceding definition indicates, type derivations of $\lambda{\to}$ have a lot in common with derivations in minimal logic. The Curry-Howard correspondence establishes a correspondence between terms and proofs and between types and logical formulas. Implication introduction corresponds to a λ-abstraction and implication elimination corresponds to an application.

$$\frac{x : \rho \in \Gamma}{\Gamma \vdash x : \rho} \qquad \frac{\Gamma, x : \rho \vdash t : \delta}{\Gamma \vdash \lambda x : \rho.t : \rho \to \delta} \qquad \frac{\Gamma \vdash t : \rho \to \delta \qquad \Gamma \vdash s : \rho}{\Gamma \vdash ts : \delta}$$

(a) var (b) lambda (c) app

Figure 2.4: The typing rules of $\lambda\!\to$.

Theorem 2.3.5. *We have $\Gamma \vdash A$ in minimal first-order logic iff $\Gamma \vdash t : A$ for some term t in $\lambda\!\to$.*

The simply typed λ-calculus has some convenient properties, we will describe some of these properties now.

Lemma 2.3.6. (Thinning) *If $\Gamma \vdash t : \rho$ and $\Delta \supseteq \Gamma$, then $\Delta \vdash t : \rho$*

Proof. By induction on the derivation $\Gamma \vdash t : \rho$. □

Lemma 2.3.7. *Typing in $\lambda\!\to$ is preserved under substitution. That is, if $\Gamma, x : \delta, \Delta \vdash t : \rho$ and $\Gamma \vdash r : \delta$, then $\Gamma, \Delta \vdash t[x := r] : \rho$.*

Proof. By induction on the derivation $\Gamma, x : \delta, \Delta \vdash t : \rho$. The only interesting case is $\Gamma, x : \delta, \Delta \vdash y : \rho$. Here we distinguish the cases $y \neq x$ and $y = x$. The former case holds by assumption because $y[x := r] \equiv y$. In the latter case we also have $\rho = \delta$, so it remains to prove that $\Gamma, \Delta \vdash x[x := r] : \delta$. But we have $\Gamma \vdash r : \delta$ by assumption, so by Lemma 2.3.6 we are done. □

Lemma 2.3.8. (Subject reduction) *If $\Gamma \vdash t : \rho$ and $t \to t'$, then $\Gamma \vdash t' : \rho$.*

Proof. This result follows from the following derivation.

$$\frac{\dfrac{\Gamma, x : \delta \vdash t : \rho}{\Gamma \vdash \lambda x.t : \delta \to \rho} \qquad \Gamma \vdash r : \delta}{\Gamma \vdash (\lambda x.t)r : \rho} \quad \to_\beta \quad \Gamma \vdash t[x := r] : \rho$$

Here we have $\Gamma \vdash t[x := r] : \rho$ by Lemma 2.3.7. □

As the previous proof indicates, a β-redex corresponds to a detour in logic and β-reduction corresponds to detour elimination.

Theorem 2.3.9. *$\lambda\!\to$ is confluent. That is, if $t_1 \twoheadrightarrow t_2$ and $t_1 \twoheadrightarrow t_3$, then there exists a term t_4 such that $t_2 \twoheadrightarrow t_4$ and $t_3 \twoheadrightarrow t_4$.*

Proof. This result follows immediately from confluence of the untyped λ-calculus (Theorem 2.2.5) and subject reduction of $\lambda\!\to$ (Lemma 2.3.8). □

Lemma 2.3.10. *$\lambda\!\to$ is* strongly normalizing. *That is, given a term t such that $\Gamma \vdash t : \rho$ for some type ρ, then each reduction path yields a normal form.*

Proof. Proven in [GTL89, Geu08, for example]. □

Contrary to the untyped λ-calculus (Theorem 2.2.6), it is decidable whether two well-typed $\lambda\!\!\rightarrow$-terms are β-equivalent. This is done by reducing both terms to a normal form, which will terminate because $\lambda\!\!\rightarrow$ is strongly normalizing (Lemma 2.3.10). Moreover, we know that that each term has a unique normal form because $\lambda\!\!\rightarrow$ is confluent (Lemma 2.3.8). Hence it is sufficient to test whether those normal forms are α-equivalent.

In $\lambda\!\!\rightarrow$, it is possible to encode most well-known data types (integers, lists, trees, *etc.*). For example, we can encode the natural numbers as the *first-order Church numerals*.

Definition 2.3.11. *The type* N_ρ *of the* first-order Church numerals over a type ρ *is defined as follows.*

$$\mathsf{N}_\rho := (\rho \to \rho) \to \rho \to \rho$$

A natural number n *is encoded by a* λ*-term as follows.*

$$c_{n,\rho} := \lambda f : \rho \to \rho . \lambda x : \rho \,.\, f^n x$$

Fact 2.3.12. *The Church numerals are well-typed. That is,* $\Gamma \vdash c_{n,\rho} : \mathsf{N}_\rho$.

An important property of the Church numerals in $\lambda\!\!\rightarrow$ is that closed normal forms of type N_α are of the shape $c_{n,\alpha}$ for some $n \in \mathbb{N}$. Hence there is a one-to-one correspondence between natural numbers and closed normal forms of type N_α. In order to prove this result we state the following auxiliary lemma.

Lemma 2.3.13. *Given a term* t *that is in normal and moreover such that* $f : \alpha \to \alpha, x : \alpha \vdash t : \rho$*, then:*

1. *If* $\rho = \alpha$*, then* $t \equiv f^n x$ *for some* $n \in \mathbb{N}$.
2. *If* $\rho = \gamma \to \delta$*, then* $t \equiv \lambda y.r$ *for a variable* y *and term* r *or* $t \equiv f$ *provided that* $\gamma = \delta = \alpha$.

Proof. By induction on the derivation $\Delta \vdash t : \rho$.

(var) Let $\Gamma \vdash y : \rho$. Now $y \equiv f$ or $y \equiv x$ by assumption, so we are done.

(λ) Let $\Gamma \vdash \lambda y.r : \gamma \to \delta$. Now we are immediately done.

(app) Let $\Gamma \vdash rs : \rho$ with $\Gamma \vdash r : \delta \to \rho$ and $\Gamma \vdash s : \delta$. Now, by the induction hypothesis, we have either $r \equiv \lambda x.r'$ or $r \equiv f$ provided that $\rho = \delta = \alpha$. In the first case we are done, because rs should be in normal form. In the latter case we also have $s \equiv f^n x$ for some $n \in \mathbb{N}$ by the induction hypothesis, hence $rs \equiv f^{n+1} x$. □

Corollary 2.3.14. *Given a closed term* t *that is in normal form and such that* $\vdash t : \mathsf{N}_\alpha$*, then* $t \equiv c_{n,\alpha}$ *for some* $n \in \mathbb{N}$.

Proof. This result follows immediately from Lemma 2.3.13. □

Now we can define the basic operations, like the successor function, addition and multiplication functions as follows.

$$\begin{aligned}
\mathsf{S}_\rho &:= \lambda n : \mathsf{N}_\rho . \lambda f : \rho \to \rho . \lambda x : \rho \,.\, f(nfx) \\
\mathsf{plus}_\rho &:= \lambda n : \mathsf{N}_\rho . \lambda m : \mathsf{N}_\rho . \lambda f : \rho \to \rho . \lambda x : \rho \;.\; nf(mfx)x \\
\mathsf{times}_\rho &:= \lambda n : \mathsf{N}_\rho . \lambda m : \mathsf{N}_\rho . \lambda f : \rho \to \rho \,.\, n(mf)
\end{aligned}$$

It is not too hard to extend this method of encoding to more interesting data types. For example, lists with elements of type δ can be encoded by the type $(\delta \to \rho \to \rho) \to \rho \to \rho$ and then a list $[r_1, \ldots r_n]$ is encoded by the term $\lambda f : \delta \to \rho \to \rho . \lambda x : \rho \,.\, f\, r_1\, (f\, r_2\, \ldots\, (f\, r_n\, x))$. Moreover, binary trees with labels of type δ can be encoded by the type $(\delta \to \rho \to \rho \to \rho) \to \rho \to \rho$, *etc.* Unfortunately, the class of definable functions on these data types is quite limited since one has to fix a type ρ for each encoding. For example, the predecessor on Church numerals cannot be defined. More specifically, the $\lambda\!\to$ definable functions are exactly the extended polynomials [SU06, Geu08].

2.4 Second-order propositional logic

In first-order propositional logic it is merely possible to say something about a single formula instead of saying something about many formulas. Second-order logic has the ability to quantify over propositions and is therefore able to say something about many formulas.

Definition 2.4.1. Second-order proposition formulas *are built from an infinite set of atoms $(X, Y, \ldots)$, an implication arrow $(\to)$ and a universal quantifier over propositions $(\forall)$.*

$$A, B ::= X \mid A \to B \mid \forall X.A$$

We adopt the usual notion of capture avoiding substitution $A[X := B]$ for second-order formulas.

Definition 2.4.2. *A judgment $\Gamma \vdash A$ is derivable in* second-order propositional logic *if it can be derived by the natural deduction rules shown in Figure 2.5.*

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ \text{(a) axiom} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ (\to_i) \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ (\to_e)$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall X.A}\ X \notin \mathrm{FV}(\Gamma)\ (\forall_i) \qquad \frac{\Gamma \vdash \forall X.A}{\Gamma \vdash A[X := B]}\ (\forall_e)$$

Figure 2.5: Natural deduction for second-order propositional logic.

For first-order propositional logic we made a distinction between minimal and intuitionistic logic. However, this distinction vanishes in second-order logic because the connective false can be defined as $\bot := \forall P.P$ and Ex Falso Quodlibet then follows from $\forall$-elimination.

Definition 2.4.3. *A judgment holds in* second-order classical logic *if it can be derived by minimal second-order natural deduction and Peirce's law.*

Similarly, the distinction between minimal classical second-order and classical second-order logic vanishes because Peirce's law and double negation are logically equivalent in second-order logic.

Lemma 2.4.4. *Peirce's law* $\forall XY.((X \to Y) \to X) \to X$ *and double negation* $\forall X.\neg\neg X \to X$ *are logically equivalent.*

Proof. Let us assume that Peirce's law holds.

$$\dfrac{\dfrac{\forall XY.((X \to Y) \to X) \to X}{(\neg X \to X) \to X} \qquad \dfrac{\dfrac{[\neg\neg X] \qquad [\neg X]}{\dfrac{\bot}{X}}}{\neg X \to X}}{\dfrac{X}{\forall X.\neg\neg X \to X}}$$

Conversely, let us assume that double negation holds.

$$\dfrac{\dfrac{\forall X.\neg\neg X \to X}{\neg\neg X \to X} \qquad \dfrac{\dfrac{[\neg X] \qquad \dfrac{[(X \to Y) \to X] \qquad \dfrac{\dfrac{[\neg X] \qquad [X]}{\bot}}{Y}}{X \to Y}}{X}}{\bot}}{\neg\neg X}}{\dfrac{X}{\forall XY.((X \to Y) \to X) \to X}}$$

□

2.5 The second-order typed λ-calculus

In this section we discuss the second-order typed λ-calculus (henceforth $\lambda^{\mathbf{2}}$), which is also known as System **F**. The $\lambda^{\mathbf{2}}$-calculus is basically an extension of the $\lambda{\to}$-calculus obtained by adding abstraction over types, which turns out to be very powerful. Abstraction over types also extends the Curry-Howard correspondence to second-order propositional logic.

As we have shown in Section 2.3, we can encode all well-known data types (integers, lists, trees, *etc.*) in $\lambda{\to}$. Unfortunately, for each encoding we had to fix a type ρ and as a result the $\lambda{\to}$ definable functions are exactly the extended polynomials. In $\lambda^{\mathbf{2}}$, we are however able to abstract over this type ρ and are able to encode all primitive recursive functions [BB85].

Definition 2.5.1. Second-order types *are built from an infinite set of type variables* $(\alpha, \beta, \ldots)$*, an implication arrow* $(\to)$ *and a universal quantifier over types* $(\forall)$*.*

$$\rho, \delta ::= \alpha \mid \rho \to \delta \mid \forall\alpha.\rho$$

Definition 2.5.2. *The* terms *of* $\lambda^{\mathbf{2}}$ *are inductively defined over an infinite set of variables* $(x, y, \ldots)$ *as follows.*

$$t, r, s ::= x \mid \lambda x : \rho.t \mid ts \mid \lambda\alpha.t \mid t\rho$$

As usual, we let $\mathrm{FV}(t)$ and $\mathrm{FTV}(t)$ denote the set of free variables, and free type variables of a term t, respectively. Moreover, the operation of capture avoiding substitution $t[x := r]$ of r for x in t and capture avoiding substitution $t[\alpha := \rho]$ of ρ for α in t generalize to $\lambda\mathbf{2}$-terms in the obvious way.

Definition 2.5.3. *A $\lambda\mathbf{2}$-typing judgment $\Gamma \vdash t : \rho$ denotes that a term t has type ρ in an environment Γ. The derivation rules for such judgments are shown in Figure 2.6.*

$$\frac{x : \rho \in \Gamma}{\Gamma \vdash x : \rho}\ \text{(a) var} \qquad \frac{\Gamma, x : \rho \vdash t : \delta}{\Gamma \vdash \lambda x : \rho.t : \rho \to \delta}\ \text{(b) lambda} \qquad \frac{\Gamma \vdash t : \rho \to \delta \qquad \Gamma \vdash s : \rho}{\Gamma \vdash ts : \delta}\ \text{(c) app}$$

$$\frac{\Gamma \vdash t : \rho}{\Gamma \vdash \lambda\alpha.t : \forall\alpha.\rho}\ \alpha \notin \mathrm{FTV}(\Gamma)\ \text{(d) } \forall_i \qquad \frac{\Gamma \vdash t : \forall\alpha.\rho}{\Gamma \vdash t\delta : \rho[\alpha := \delta]}\ \text{(e) } \forall_e$$

Figure 2.6: The typing rules of $\lambda\mathbf{2}$.

Now we can extend the Curry-Howard correspondence to second-order logic: $\forall$-introduction corresponds to a λ-abstraction over types and $\forall$-elimination corresponds to an application of a term to a type.

Theorem 2.5.4. *We have $\Gamma \vdash A$ in minimal second-order logic iff $\Gamma \vdash t : A$ for some term t in $\lambda\mathbf{2}$.*

In Section 2.3, we did not explicitly define the notion of β-reduction for $\lambda{\to}$-terms. However, in $\lambda\mathbf{2}$ we have two kinds of lambdas, one to abstract over terms and one to abstract over types. Therefore we define the notion of β-reduction for $\lambda\mathbf{2}$-terms explicitly.

Definition 2.5.5. *Reduction $t \to t'$ on $\lambda\mathbf{2}$-terms is defined as the compatible closure of the following rules.*

$$\begin{aligned} (\lambda x : \rho.t)r &\to_\beta t[x := r] \\ (\lambda\alpha.t)\rho &\to_{\beta\forall} t[\alpha := \rho] \end{aligned}$$

As usual, $\twoheadrightarrow$ denotes the reflexive/transitive closure and $=$ denotes the reflexive/symmetric/transitive closure.

Just like the $\lambda{\to}$-calculus, the $\lambda\mathbf{2}$-calculus is confluent, satisfies subject reduction and is strongly normalizing [Bar92, SU06, Geu08].

In the preceding section, we have defined the natural numbers as first-order *Church numerals* over a fixed type ρ as $(\rho \to \rho) \to \rho \to \rho$. However, abstraction over types allows us to do this polymorphically in $\lambda\mathbf{2}$.

Definition 2.5.6. *The type* N *of the* second-order Church numerals *is defined as follows.*

$$\mathsf{N} := \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha$$

A natural number n is encoded by a $\lambda\mathbf{2}$-term c_n as follows.

$$c_n := \lambda\alpha\lambda f : \alpha \to \alpha.\lambda x : \alpha\,.\, f^n x$$

Fact 2.5.7. *The Church numerals are well-typed. That is,* $\Gamma \vdash c_n : \mathsf{N}$.

In the same way as we have proven for $\lambda\!\!\rightarrow$ we can prove that there is a one-to-one correspondence between natural numbers and closed normals forms of type N in $\lambda^{\mathbf{2}}$.

Lemma 2.5.8. *Given a closed term* t *that is in normal form and such that* $\vdash t : \mathsf{N}$*, then* $t \equiv c_n$ *for some* $n \in \mathbb{N}$.

Proof. Similarly to the proof of Corollary 2.3.14. □

Now we define the basic operations, like the successor function, addition and multiplication functions as follows.

$$\begin{aligned}
\mathsf{S} &:= \lambda n : \mathsf{N}.\lambda\alpha\lambda f : \alpha \to \alpha.\lambda x : \alpha \,.\, f(n\alpha f x) \\
\mathsf{plus} &:= \lambda n : \mathsf{N}.\lambda m : \mathsf{N}.\lambda\alpha.\lambda f : \alpha \to \alpha.\lambda x : \alpha \,.\, n\alpha f(m\alpha f x)x \\
\mathsf{times} &:= \lambda n : \mathsf{N}.\lambda m : \mathsf{N}.\lambda\alpha.\lambda f : \alpha \to \alpha \,.\, n\alpha(m\alpha f)
\end{aligned}$$

In order to show that $\lambda^{\mathbf{2}}$ has more expressive power than $\lambda\!\!\rightarrow$, we will encode primitive recursion on Church-numerals. Fortunately, primitive recursion can be encoded in terms of iteration and products. As one could see, the Church numerals basically include iteration, so we get that for free. That leaves us to encode products.

Definition 2.5.9. *The* product type $\rho_1 \times \rho_2$ *is defined as follows.*

$$\rho_1 \times \rho_2 := \forall\alpha.(\rho_1 \to \rho_2 \to \alpha) \to \alpha$$

The pair $\langle t_1, t_2\rangle$ *of the* $\lambda^{\mathbf{2}}$*-terms* $t_1 : \rho_1$ *and* $t_2 : \rho_1$*, and* the projections $\pi_i r$ *of a product* $r : \rho_1 \times \rho_2$*, are defined as follows.*

$$\begin{aligned}
\langle t_1, t_2\rangle &:= \lambda\alpha.\lambda f : \rho_1 \to \rho_2 \to \alpha \,.\, f\, t_1\, t_2 \\
\pi_i r &:= r\, \rho_i\, (\lambda x_1 : \rho_1.\lambda x_2 : \rho_2 \,.\, x_i)
\end{aligned}$$

Fact 2.5.10. *Pairs and projections are well-typed. That is:*

1. *Given terms* $t_1 : \rho_1$ *and* $t_2 : \rho_2$*, then* $\langle t_1, t_2\rangle : \rho_1 \times \rho_2$.
2. *Given a term* $r : \rho_1 \times \rho_2$*, then* $\pi_i r : \rho_i$.

Lemma 2.5.11. *Projections are defined correctly. That is,* $\pi_i\langle t_1, t_2\rangle \twoheadrightarrow t_i$.

Proof. Let $t_1 : \rho_1$ and $t_2 : \rho_1$. Now:

$$\begin{aligned}
\pi_i\langle t_1, t_2\rangle &\equiv (\lambda\alpha.\lambda f : \rho_1 \to \rho_2 \to \alpha.f\, t_1\, t_2)\, \rho_i\, (\lambda x_1 : \rho_1.\lambda x_2 \,.\, x_i) \\
&\twoheadrightarrow (\lambda x_1 x_2.x_i)\, t_1\, t_2 \\
&\twoheadrightarrow t_i
\end{aligned}$$

□

Now we define primitive recursion in the ordinary way.

Definition 2.5.12. *Given terms* $r : \rho$*,* $s : \mathsf{N} \to \rho \to \rho$ *and* $t : \mathsf{N}$*, then* primitive recursion $\mathsf{nrec}_\rho\ r\ s\ t$ *is defined as follows.*

$$\mathsf{nrec}_\rho\ r\ s\ t := \pi_1(t\ (\rho \times \mathsf{N})\ (\lambda h.\langle s\ (\pi_2 h)\ (\pi_1 h), \mathsf{S}(\pi_2 h)\rangle)\ \langle r, c_0\rangle)$$

Fact 2.5.13. *Primitive recursion is well-typed. That is, given terms $t : \mathsf{N}$, $r : \rho$ and $s : \mathsf{N} \to \rho \to \rho$, then $\mathsf{nrec}_\rho\ r\ s\ t : \rho$.*

Lemma 2.5.14. *Primitive recursion is defined correctly. That is, given terms $r : \rho$ and $s : \mathsf{N} \to \rho \to \rho$, then:*

$$\begin{aligned}\mathsf{nrec}_\rho\ r\ s\ c_0 &= r\\ \mathsf{nrec}_\rho\ r\ s\ c_{n+1} &= s\ c_n\ (\mathsf{nrec}_\rho\ r\ s\ c_n)\end{aligned}$$

Proof. In order to prove this result we have to generalize it slightly. Given terms $r : \rho$, $s : \mathsf{N} \to \rho \to \rho$ and $t : \mathsf{N}$, define:

$$\overline{\mathsf{nrec}}_\rho\ r\ s\ t := t\ (\rho \times \mathsf{N})\ f'\ x'$$

where $f' := (\lambda h.\langle s\ (\pi_2 h)\ (\pi_1 h), \mathsf{S}(\pi_2 h)\rangle)$ and $x' := \langle r, c_0\rangle$. Now we also prove:

$$\overline{\mathsf{nrec}}_\rho\ r\ s\ c_n = \langle \mathsf{nrec}_\rho\ r\ s\ c_n, c_n\rangle$$

We proceed by induction on n.

1. Suppose that $n = 0$. Now we have the following.

$$\begin{aligned}\overline{\mathsf{nrec}}_\rho\ r\ s\ c_0 &\equiv (\lambda\alpha\lambda f\lambda x.x)\ (\rho \times \mathsf{N})\ f'\ x'\\ &\twoheadrightarrow x'\\ &\equiv \langle r, c_0\rangle\end{aligned}$$

   So we also have $\mathsf{nrec}_\rho\ r\ s\ c_0 = r$.

2. Suppose that $n > 0$. By the induction hypothesis:

$$\begin{aligned}\langle \mathsf{nrec}_\rho\ r\ s\ c_n, c_n\rangle &= \overline{\mathsf{nrec}}_\rho\ r\ s\ c_n\\ &\equiv (\lambda\alpha\lambda f\lambda x.f^n x)\ f'\ x'\\ &\twoheadrightarrow f'^n x'\end{aligned}$$

   Now we have the following.

$$\begin{aligned}\overline{\mathsf{nrec}}_\rho\ r\ s\ c_{n+1} &\equiv (\lambda\alpha\lambda f\lambda x.f^{n+1}x)\ (\rho \times \mathsf{N})\ f'\ x'\\ &\twoheadrightarrow (\lambda h.\langle s\ (\pi_1 h)\ (\pi_2 h), \mathsf{S}(\pi_2 h)\rangle)\ (f'^n x')\\ &\to \langle s\ (\pi_1(f'^n x'))\ (\pi_2(f'^n x')), \mathsf{S}(\pi_2(f'^n x'))\rangle)\\ &= \langle s\ c_n\ (\mathsf{nrec}_\rho\ r\ s\ c_n), \mathsf{S}c_n\rangle\end{aligned}$$

   So we also have $\mathsf{nrec}_\rho\ r\ s\ c_{n+1} = s\ c_n\ (\mathsf{nrec}_\rho\ r\ s\ c_n)$. □

By primitive recursion we can define functions that were impossible to define in $\lambda\!\to$. For example, the predecessor: $\mathtt{pred} := \lambda z.\mathsf{nrec}\ 0\ (\lambda xy.x)\ z$.

We can again extend this method of encoding to more interesting data types. For example, lists with elements of type δ can be encoded by the type $\forall\alpha.(\delta \to \alpha \to \alpha) \to \alpha \to \alpha$, binary trees with labels of type δ by $\forall\alpha.(\delta \to \alpha \to \alpha \to \alpha) \to \alpha \to \alpha$, *etc.* For an extensive discussion of these encodings we refer to [BB85].

Chapter 3

Classical logic and control operators

In this chapter we will investigate various extensions of the λ-calculus that support control mechanisms. In Section 3.1 we discuss Felleisen [FF86] and Griffin's [Gri90] pioneering work in this field. The former was the first to define an extension of the λ-calculus with control, while the latter was the first to find out that typed λ-calculi with control operators extend the Curry-Howard isomorphism to classical logic.

Unfortunately, Felleisen's calculus is "not pure". That is, some rules are only allowed to be applied at the top-level rather than at arbitrary places. Because of this defect it is very hard to reason equationally about programs in their calculus. In order to repair this issue Felleisen and Hieb introduced the revised theory of control [FH92], but this theory still suffers from various weaknesses [AH08]. Therefore we will look at two alternatives: the λ_Δ-calculus by Rehof and Sørensen [RS94] in Section 3.2 and the λ_μ-calculus by Parigot [Par92] in Section 3.3.

Finally, in Section 3.5 we will discuss CPS-translations, a way to simulate control in a system without it.

3.1 The $\lambda_{\mathcal{C}}$-calculus

In this section we present the $\lambda_{\mathcal{C}}$-calculus by Felleisen *et al.* [FF86, FFKD87] and the corresponding typing rules by Griffin [Gri90]. Before we present the typed $\lambda_{\mathcal{C}}$-calculus, we present its untyped version, which basically extends the λ_v-calculus with unary operators $\mathcal{C}$ and $\mathcal{A}$.

Definition 3.1.1. Untyped $\lambda_{\mathcal{C}}$-terms *are inductively defined over an infinite set of variables* $(x, y, \ldots)$ *as follows.*

$$t, r, s ::= x \mid \lambda x.t \mid ts \mid \mathcal{C}t \mid \mathcal{A}t$$

The control operators $\mathcal{A}$ and $\mathcal{C}$ abort the current evaluation context and resume in an empty evaluation context. Whereas the operator $\mathcal{A}$ discards the

current context, the operator $\mathcal{C}$ stores it for invocation at a later time. This behavior is best described by the following evaluation strategy.

Definition 3.1.2. Call-by-value evaluation $t \rhd t'$ *on terms* t *and* t' *is defined as the union of the following rules.*

$$\begin{array}{rll} E[(\lambda x.t)v] & \rhd_{\beta_v} & E[t[x := v]] \\ E[\mathcal{A}t] & \rhd_{\mathcal{A}} & t \\ E[\mathcal{C}t] & \rhd_{\mathcal{C}} & t(\lambda x.\mathcal{A}E[x]) \end{array}$$

Here, E *is a call-by-value evaluation context (Definition 2.2.13).*

Again we have to verify that we have indeed specified a deterministic evaluation strategy.

Lemma 3.1.3. *Each closed* λ*-term is either a value or can be written uniquely as* $E[(\lambda x.t)v]$*,* $E[\mathcal{A}t]$ *or* $E[\mathcal{C}t]$ *where* E *is a call-by-value evaluation context.*

Whenever the $\rhd_{\mathcal{C}}$-rule is applied, it stores the current evaluation context E as $\lambda x.\mathcal{A}E[x]$. At a later moment the program is able to invoke the term $\lambda x.\mathcal{A}E[x]$ and thereby jump back to the evaluation context E. It is essential that the operator $\mathcal{A}$ is present, because on invocation of $\lambda x.\mathcal{A}E[x]$, it makes sure that the current context is discarded and that the program jumps back to the required evaluation context.

Using these control operators we can mimic the control operators `catch` and `throw` in the $\lambda_{\mathcal{C}}$-calculus as follows.

$$\begin{array}{l} \texttt{catch}\ k\ t := \mathcal{C}(\lambda k.kt) \\ \texttt{throw}\ k\ t := kt \end{array}$$

Let us consider the term $E[\texttt{catch}\ k\ t]$. This term evaluates as follows.

$$\begin{array}{rl} E[\texttt{catch}\ k\ t] & \rhd (\lambda k.kt)(\lambda x.\mathcal{A}E[x]) \\ & \rhd (\lambda x.\mathcal{A}E[x])(t[k := \lambda x.\mathcal{A}E[x]]) \\ & \rhd \mathcal{A}E[t[k := \lambda x.\mathcal{A}E[x]]] \\ & \rhd E[t[k := \lambda x.\mathcal{A}E[x]]] \end{array}$$

Now, the term $\lambda x.\mathcal{A}E[x]$ is substituted for all occurrences of the variable k in t. Hence each subterm $\texttt{throw}\ k\ s$ becomes $(\lambda x.\mathcal{A}E[x])s$. Let us see what happens when such a subterm is evaluated.

$$\begin{array}{rl} F[(\lambda x.\mathcal{A}E[x])s] & \rhd F[\mathcal{A}E[s]] \\ & \rhd E[s] \end{array}$$

As shown, the evaluation context F is discarded and the program resumes in the context E, which is the context that we have stored during evaluation of the corresponding `catch`. Hence, this simulation mimics the behavior of `catch` and `throw` as we have described in Section 1.1.

It is interesting to remark that the control operator $\mathcal{A}$ is superfluous since it can be simulated by $\mathcal{C}$.

Lemma 3.1.4. *The control operator $\mathcal{A}$ can be simulated by $\mathcal{C}$ as follows.*

$$\mathcal{A}t := \mathcal{C}(\lambda k.t) \text{ provided that } k \notin \mathrm{FV}(t)$$

Proof. $E[\mathcal{A}t] \equiv E[\mathcal{C}(\lambda k.t)] \rhd (\lambda k.t)\,(\lambda x.\mathcal{A}E[x]) \rhd t.$ □

For a long time people thought that the Curry-Howard correspondence was limited to minimal logic. But in 1990 Griffin [Gri90] discovered that the Curry-Howard correspondence can be extended to classical logic by incorporation of control operators. First we extend simple types with a basic type $\bot$ and present $\lambda_{\mathcal{C}}$-terms in Church style.

Definition 3.1.5. Intuitionistic types *are built from an infinite set of type variables ($\alpha, \beta, \ldots$), a basic type ($\bot$) and an implication arrow ($\rightarrow$).*

$$\rho, \delta ::= \bot \mid \alpha \mid \rho \rightarrow \delta$$

Definition 3.1.6. *The* terms *of $\lambda_{\mathcal{C}}\!\rightarrow$ are inductively defined over an infinite set of variables $(x, y, \ldots)$ as follows.*

$$t, r, s ::= x \mid \lambda x : \rho.t \mid ts \mid \mathcal{C}_\rho t \mid \mathcal{A}_\rho t$$

Here, ρ ranges over intuitionistic types.

Definition 3.1.7. *A $\lambda_{\mathcal{C}}\!\rightarrow$-*typing judgment *$\Gamma \vdash t : \rho$ denotes that a term t has type ρ in an environment Γ. The derivation rules for such judgments are shown in Figure 3.1.*

$$\frac{x : \rho \in \Gamma}{\Gamma \vdash x : \rho}\ \text{(a) var} \qquad \frac{\Gamma, x : \rho \vdash t : \delta}{\Gamma \vdash \lambda x : \rho.t : \rho \rightarrow \delta}\ \text{(b) lambda} \qquad \frac{\Gamma \vdash t : \rho \rightarrow \delta \qquad \Gamma \vdash s : \rho}{\Gamma \vdash ts : \delta}\ \text{(c) app}$$

$$\frac{\Gamma \vdash t : \bot}{\Gamma \vdash \mathcal{A}_\rho t : \rho}\ \text{(d) abort} \qquad \frac{\Gamma \vdash t : \neg\neg\rho}{\Gamma \vdash \mathcal{C}_\rho t : \rho}\ \text{(e) control}$$

Figure 3.1: The typing rules of $\lambda_{\mathcal{C}}\!\rightarrow$.

Definition 3.1.8. *A $\lambda_{\mathcal{A}}\!\rightarrow$-*typing judgment *$\Gamma \vdash t : \rho$ denotes that a term t has type ρ in an environment Γ. The derivation rules for such judgments are shown in Figure 3.1 (a-d).*

Lemma 3.1.9. *The simulation of the operator $\mathcal{A}$ by $\mathcal{C}$ is well-typed.*

Proof. Let $t : \bot$, now we have $\mathcal{C}_\rho(\lambda k : \neg\rho.t) : \rho$ as shown below.

$$\frac{\dfrac{t : \bot}{\lambda k : \neg\rho.t : \neg\neg\rho}}{\mathcal{C}_\rho(\lambda k : \neg\rho.t) : \rho}$$

□

Now we can extend the Curry-Howard correspondence to classical logic: the abort rule corresponds to Ex Falso Quodlibet and the control rule corresponds to double negation.

Theorem 3.1.10. *We have $\Gamma \vdash A$ in classical logic (respectively intuitionistic logic) iff $\Gamma \vdash t : A$ for some term t in $\lambda_{\mathcal{C}}\!\rightarrow$ (respectively $\lambda_{\mathcal{A}}\!\rightarrow$).*

Unfortunately, the simulation of `catch` and `throw`, which we have described previously, is not well-typed. If we consider `throw` $k\ t : \rho$ with $t : \delta$, then `throw` $k\ t$ should be a type cast from δ to ρ. However, δ has to be equal to $\bot$ because $k : \neg\delta$. Luckily, this problem can easily be repaired.

$$\texttt{throw}\ k\ t := \mathcal{A}(kt)$$

By addition of the control operator $\mathcal{A}$ we have turned the `throw` into an actual type cast from δ to ρ, so it remains to check that it still mimics the behavior of `throw`. Let us consider the term $F[\texttt{throw}\ k\ s]$ with $\lambda x.\mathcal{A}E[x]$ substituted for k. This term evaluates as follows.

$$\begin{aligned} F[\mathcal{A}((\lambda x.\mathcal{A}E[x])s)] &\rhd (\lambda x.\mathcal{A}E[x])s \\ &\rhd \mathcal{A}E[s] \\ &\rhd E[s] \end{aligned}$$

Now that we have shown that this system is capable of simulating `catch` and `throw` we will take a look at some meta theoretical properties. First we will consider subject reduction, so let us see what happens if we try to prove that the $\rhd_{\mathcal{A}}$-rule preserves typing.

$$\dfrac{\dfrac{t : \bot}{\mathcal{A}_\rho t : \rho}}{E[\mathcal{A}_\rho t] : \delta} \quad \rhd_{\mathcal{A}} \quad t : \bot$$

Here, $E[\mathcal{A}_\rho t]$ is supposed to have type $\bot$. However, closed terms of type $\bot$ do not exist by Theorem 3.1.10 and consistency of classical logic [SU06]. In order to repair this issue, Griffin altered the evaluation rules [Gri90]. In his system each term is surrounded by $\mathcal{C}(\lambda k.k\_)$ and evaluation is only allowed within $\mathcal{C}(\lambda k.\_)$. The evaluation rules are as follows.

Definition 3.1.11. Evaluation à la Griffin $t \rhd t'$ *on terms t and t' is defined as the union of the following rules.*

$$\begin{aligned} \mathcal{C}(\lambda k.E[(\lambda x.t)v]) &\rhd_{t\beta_v} \mathcal{C}(\lambda k.E[t[x := v]]) \\ \mathcal{C}(\lambda k.E[\mathcal{A}t]) &\rhd_{t\mathcal{A}} \mathcal{C}(\lambda k.t) \\ \mathcal{C}(\lambda k.E[\mathcal{C}t]) &\rhd_{t\mathcal{C}} \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}E[x]) \\ \mathcal{C}(\lambda k.kv) &\rhd_{\mathcal{C}e} v \qquad \textit{provided } k \notin \mathrm{FV}(v) \end{aligned}$$

Lemma 3.1.12. *$\lambda_{\mathcal{C}}\!\rightarrow$ à la Griffin satisfies subject reduction.*

Proof. First we have to prove a similar substitution lemma as we have proven for $\lambda\!\rightarrow$ (Lemma 2.3.7). Then we have to prove that all evaluation rules preserve typing. We treat some interesting evaluation rules.

1. The $\rhd_{t\mathcal{C}}$-rule:

$$\dfrac{\dfrac{\dfrac{t : \neg\neg\rho}{\mathcal{C}t : \rho}}{\overset{=}{E}[\mathcal{C}t] : \bot}}{\dfrac{\lambda k.E[\mathcal{C}t] : \neg\neg\delta}{\mathcal{C}(\lambda k.E[\mathcal{C}t]) : \delta}} \quad \rhd_{t\mathcal{C}} \quad \dfrac{t : \neg\neg\rho \qquad \dfrac{\dfrac{\overset{=}{x} : \rho}{\overset{=}{E}[x] : \bot}}{\mathcal{A}E[x] : \bot}}{\dfrac{t\lambda x.\mathcal{A}E[x] : \bot}{\dfrac{\lambda k.t\lambda x.\mathcal{A}E[x] : \neg\neg\delta}{\mathcal{C}(\lambda k.t\lambda x.\mathcal{A}E[x]) : \delta}}}$$

2. The $\rhd_{\mathcal{C}e}$-rule:

$$\dfrac{\dfrac{\dfrac{k : \neg\rho \qquad v : \rho}{kv : \bot}}{\lambda k.kv : \neg\neg\rho}}{\mathcal{C}(\lambda k.kv) : \rho} \quad \rhd_{\mathcal{C}e} \quad v : \rho$$

□

Lemma 3.1.13. *$\lambda_{\mathcal{C}}\!\rightarrow$ à la Griffin is strongly normalizing.*

Proof. Proven in [Gri90]. □

The $\lambda_{\mathcal{C}}$-calculus, as presented here, is not a well-suited framework for reasoning about programs, because if we have $r_1 \rhd^* r_2$ we do not necessarily have $E[r_1] \rhd^* E[r_2]$. For example, let $E \neq \Box$, now we have $(\mathcal{A}\mathsf{I}) \rhd \mathsf{I}$ and $E[\mathcal{A}\mathsf{I}] \rhd \mathcal{A}\mathsf{I} \not\rhd^* \mathsf{E[I]}$. So, if we prove some property of a program r, that property does not necessarily hold if we "plug" r into another program.

Various people tried to solve this problem problem by specifying a reduction theory instead of an evaluation strategy. Unfortunately, none of the known reduction theories is able to express the semantics in way that is adequate with respect to the $\rhd_{\mathcal{C}}$-rule [AH08]. The original reduction rules by Felleisen *et al.* [FFKD87] are as follows.

$$\begin{array}{rll}
(\lambda x.t)r & \rightarrow_{\beta_v} & t[x := r] \\
(\mathcal{C}t)s & \rightarrow_{\mathcal{C}L} & \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}(k(xt))) \\
v(\mathcal{C}t) & \rightarrow_{\mathcal{C}R} & \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}(k(vx))) \\
\mathcal{C}t & \rhd_{\mathcal{C}T} & t(\lambda x.\mathcal{A}x)
\end{array}$$

The reduction rules $\rightarrow_{\mathcal{C}L}$ and $\rightarrow_{\mathcal{C}R}$ basically lift an occurrence of the $\mathcal{C}$-operator step by step to the top-level. Once it has reached the top-level, the $\rhd_{\mathcal{C}T}$-rule aborts the current continuation [AH08]. However, these rules are not satisfactory, because the $\rhd_{\mathcal{C}T}$-rule may only be applied at the top-level and moreover does not preserve typing. Felleisen and Hieb introduced the revised theory of control so as to repair these problems [FH92]. They tried to mimic the behavior of $\rhd_{\mathcal{C}T}$ by the reduction rules $\rightarrow_{\mathcal{C}t}$ and $\rightarrow_{\mathcal{C}i}$.

$$\begin{array}{rll}
(\lambda x.t)r & \rightarrow_{\beta_v} & t[x := r] \\
(\mathcal{C}t)s & \rightarrow_{\mathcal{C}L} & \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}(k(xt))) \\
v(\mathcal{C}t) & \rightarrow_{\mathcal{C}R} & \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}(k(vx))) \\
\mathcal{C}t & \rightarrow_{\mathcal{C}t} & \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}(kx)) \\
\mathcal{C}(\lambda k.\mathcal{C}t) & \rightarrow_{\mathcal{C}i} & \mathcal{C}(\lambda k.t\lambda x.\mathcal{A}x)
\end{array}$$

Unfortunately, their theory gives rise to other problems. Firstly, we can never reduce a term of the shape $\mathcal{C}t$ to a value, simply because there is no rule that allows to get rid of a $\mathcal{C}$-operator at the top-level. Secondly, the $\to_{\mathcal{C}t}$-rule can be applied infinitely many times, so this theory is (even in a simply-typed system) not strongly normalizing. For an extensive discussion of the problems related to this system we refer to [AH08].

3.2 The λ_Δ-calculus

In this section we will discuss the λ_Δ-calculus by Rehof and Sørensen [RS94]. The λ_Δ-calculus extends ordinary λ-terms with a binder Δ, which is typed by Reduction Ad Absurdum (RAA). Contrary to the $\lambda_\mathcal{C}$-calculus, which we have discussed in the preceding chapter, this system is call-by-name and satisfies the main meta theoretical properties.

Definition 3.2.1. *The* terms of λ_Δ *are inductively defined over an infinite set of variables* $(x, y, \ldots)$ *as follows.*

$$t, r, s ::= x \mid \lambda x : \rho.t \mid ts \mid \Delta x : \rho.t$$

Here, ρ *ranges over intuitionistic types (Definition 3.1.5).*

Definition 3.2.2. *A* λ_Δ-typing judgment $\Gamma \vdash t : \rho$ *denotes that a term* t *has type* ρ *in an environment* Γ*. The derivation rules for such judgments are shown in Figure 3.2.*

$$\frac{x : \rho \in \Gamma}{\Gamma \vdash x : \rho}\ \text{(a) var} \qquad \frac{\Gamma, x : \rho \vdash t : \delta}{\Gamma \vdash \lambda x : \rho.t : \rho \to \delta}\ \text{(b) lambda}$$

$$\frac{\Gamma \vdash t : \rho \to \delta \qquad \Gamma \vdash s : \rho}{\Gamma \vdash ts : \delta}\ \text{(c) app} \qquad \frac{\Gamma, x : \rho \to \bot \vdash t : \bot}{\Gamma \vdash \Delta x : \rho \to \bot.t : \rho}\ \text{(d) delta}$$

Figure 3.2: The typing rules of λ_Δ.

Theorem 3.2.3. $\Gamma \vdash A$ *in classical logic iff* $\Gamma \vdash t : A$ *for some term* t *in* λ_Δ*.*

Proof. Reduction Ad Absurdum is provable in classical logic.

$$\dfrac{\dfrac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash \neg\neg A}}{\Gamma \vdash A}$$

For the reverse implication, we prove that double negation is inhabited in λ_Δ. That is, for each term t of type $\neg\neg\rho$ we can construct a term of type ρ.

$$\dfrac{\dfrac{\Gamma, x : \neg\rho \vdash t : \neg\neg\rho \qquad \Gamma, x : \neg\rho \vdash x : \neg\rho}{\Gamma, x : \neg\rho \vdash t : \bot}}{\Gamma \vdash \Delta x.tx : \rho}$$

□

Definition 3.2.4. *Reduction $t \to t'$ on λ_Δ-terms t and t' is defined as the compatible closure of the following rules.*

$$\begin{array}{rll} (\lambda x.t)r & \to_\beta & t[x := r] \\ (\Delta x.t)s & \to_{\Delta R} & \Delta x.t[x := \lambda k.x(ks)] \\ \Delta x.xt & \to_{\Delta\eta} & t \qquad \textit{provided } x \notin \mathrm{FV}(t) \\ \Delta x.x\Delta y.t & \to_{\Delta i} & \Delta x.t[y := x] \end{array}$$

As usual, $\twoheadrightarrow$ denotes the reflexive/transitive closure and $=$ denotes the reflexive/symmetric/transitive closure.

Before we discuss the computation contents of λ_Δ, we introduce the following notation.

Notation 3.2.5. $\nabla t := \Delta y.t$ *provided that* $y \notin \mathrm{FV}(t)$.

One should think of $\Delta x.yt$ as a combined catch and throw clause: it catches "exceptions" named x in t and finally throws the result of t to y. Hence, we can mimic `catch` and `throw` as follows.

Definition 3.2.6. *The terms* `catch` $k\ t$ *and* `throw` $k\ t$ *are defined as follows.*

$$\begin{aligned} &\texttt{catch}\ k\ t := \Delta k.kt \\ &\texttt{throw}\ k\ t := \nabla(kt) \end{aligned}$$

Lemma 3.2.7. *We have the following reductions for* `catch` *and* `throw`.

1. $(\texttt{throw}\ k\ t)\ \vec{r} \twoheadrightarrow \texttt{throw}\ k\ t$
2. $\texttt{catch}\ k\ (\texttt{throw}\ k\ t) \twoheadrightarrow \texttt{catch}\ k\ t$
3. $\texttt{catch}\ k\ t \twoheadrightarrow t$ *provided that* $k \notin \mathrm{FV}(t)$

Proof. These reductions follow directly from the reduction rules of λ_Δ, except for the first one, where an induction on the length of $\vec{r}$ is needed. □

Note that we do not always have the reduction $\texttt{catch}\ k\ (\texttt{throw}\ k\ t) \twoheadrightarrow t$, because t might also contain another `throw` to k.

Keeping the preceding simulation of `catch` and `throw` in mind, it should be clear that λ_Δ is call-by-name. If we consider the term $f\,(\texttt{throw}\ k\ t)$, the `throw` has to propagate all the way through f. Also, if f does not use its argument, the `throw` will simply be ignored.

Just like the $\lambda{\to}$-calculus, the λ_Δ-calculus satisfies the main meta theoretical properties. We state the most important properties now.

Lemma 3.2.8. λ_Δ *is confluent.*

Proof. This is proven in [RS94]. □

Lemma 3.2.9. λ_Δ *satisfies subject reduction.*

Proof. First we have to prove a similar substitution lemma as we have proven for $\lambda{\to}$ (Lemma 2.3.7). Then we have to prove that all reduction rules preserve typing. We treat some interesting reduction rules.

1. The $\to_{\Delta R}$-rule:

$$\dfrac{\dfrac{t : \bot}{\Delta x.t : \rho \to \delta} \qquad s : \rho}{(\Delta x.t)s : \delta} \quad \to_{\Delta R} \quad \dfrac{t[x := \lambda k.x(ks)] : \bot}{\Delta x.t[x := \lambda k.x(ks)] : \delta}$$

   Here we have $t[x := \lambda k.x(ks)] : \bot$ by the substitution lemma and the following derivation.

$$\dfrac{\dfrac{x : \neg\delta \qquad \dfrac{k : \rho \to \delta \qquad s : \rho}{ks : \delta}}{x(ks) : \bot}}{\lambda k.x(ks) : \neg(\rho \to \delta)}$$

2. The $\to_{\Delta\eta}$-rule:

$$\dfrac{\dfrac{x : \neg\rho \qquad t : \rho}{xt : \bot}}{\Delta x.xt : \rho} \quad \to_{\Delta\eta} \quad t : \rho$$

3. The $\to_{\Delta i}$-rule:

$$\dfrac{\dfrac{x : \neg\rho \qquad \dfrac{t : \bot}{\Delta y.t : \rho}}{x\Delta y.t : \bot}}{\Delta x.x\Delta y.t : \rho} \quad \to_{\Delta i} \quad \dfrac{t[y := x] : \bot}{\Delta x.t[y := x] : \rho}$$

   Here we have $t[y := x] : \bot$ by the substitution lemma. □

Lemma 3.2.10. *λ_Δ is strongly normalizing.*

Proof. This is proven in [RS94]. □

In this section we have described the λ_Δ-calculus. This system is presented by a reduction theory and satisfies the main meta theoretical properties. Also, it seems like it is able to simulate the operators `catch` and `throw`. However, Hugo Herbelin (private communication) observed some problems.

Firstly, the system is unable to get rid of consecutive Δ-abstractions, e.g. in $\Delta x.\Delta y.t$. This is particularly troublesome if we add other data types to the system. For example, let us consider λ_Δ extended with a type `N` for natural numbers. In this system it would be satisfactory if closed normal forms of type `N` are of the shape $\underline{n}$, where $\underline{n}$ is the numeral representing some $n \in \mathbb{N}$ (à la Corollary 2.3.14 for $\lambda\to$). However, this property fails for the term $\Delta x.\Delta y.x\,\underline{0}$. Let us take a look at this term's type derivation.

$$\dfrac{\dfrac{\dfrac{x : \mathtt{N} \to \bot \qquad \underline{0} : \mathtt{N}}{x\,\underline{0} : \bot}}{\Delta y : \bot \to \bot\,.\,x\,\underline{0} : \bot}}{\Delta x : \mathtt{N} \to \bot.\Delta y : \bot \to \bot\,.\,x\,\underline{0} : \mathtt{N}}$$

From a logical point of view we observe that Reduction Ad Absurdum is applied twice. Of course, that has little use since it introduces an assumption $\bot \to \bot$. Hence it would be desirable to avoid such derivations.

Secondly, let us consider $\texttt{throw}\ k\ (\texttt{throw}\ l\ s) \equiv \Delta x.k\Delta y.ls$. One would expect that this term reduces to $(\texttt{throw}\ l\ s)$. However, because λ_Δ does not distinguish ordinary variables from continuation variables, there is no way to determine whether k is a continuation variable for which the reduction $\Delta x.k\Delta y.t \to \Delta x.t[y := k]$ should be allowed or k is an ordinary variable for which reduction should be disallowed.

3.3 The λ_μ-calculus

In this section we present the λ_μ-calculus by Parigot [Par92]. This system is quite similar to λ_Δ, however, it distinguishes ordinary variables from continuation variables and the terms are of a more restricted shape.

Definition 3.3.1. *The* terms *and* commands *of λ_μ are mutually inductively defined over an infinite set of* λ-variables $(x, y, \ldots)$ *and* μ-variables $(\alpha, \beta, \ldots)$ *as follows.*

$$\begin{aligned} t, r, s &::= x \mid \lambda x : \rho.r \mid ts \mid \mu\alpha : \rho.c \\ c, d &::= [\alpha]t \end{aligned}$$

Here, ρ ranges over simple types (Definition 2.3.1).

Remark 3.3.2. *The binding power of $[\alpha]t$ is weaker than sr, so instead of $[\alpha](sr)$, we just write $[\alpha]sr$.*

As usual, we let $\mathrm{FV}(t)$ and $\mathrm{FCV}(t)$ denote the set of free λ-variables and μ-variables of a term t, respectively. Moreover, $t[x := r]$ denotes substitution of r for x in t, which is capture avoiding for both λ- and μ-variables.

Definition 3.3.3. *A λ_μ-typing judgment $\Gamma; \Delta \vdash t : \rho$ denotes that a term t has type ρ in an environment of λ-variables Γ and an environment of μ-variables Δ. A typing judgment $\Gamma; \Delta \vdash c : \perp\!\!\!\perp$ denotes that a command c is typable in an environment of λ-variables Γ and an environment of μ-variables Δ. The derivation rules for such judgments are mutually recursively defined and shown in Figure 3.3.*

$$\frac{x : \rho \in \Gamma}{\Gamma; \Delta \vdash x : \rho}\ \text{(a) axiom} \qquad \frac{\Gamma, x : \rho; \Delta \vdash t : \delta}{\Gamma; \Delta \vdash \lambda x : \rho.t : \rho \to \delta}\ \text{(b) lambda} \qquad \frac{\Gamma; \Delta \vdash t : \rho \to \delta \qquad \Gamma; \Delta \vdash s : \rho}{\Gamma; \Delta \vdash ts : \delta}\ \text{(c) app}$$

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash c : \perp\!\!\!\perp}{\Gamma; \Delta \vdash \mu\alpha : \rho.c : \rho}\ \text{(d) activate} \qquad \frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash [\alpha]t : \perp\!\!\!\perp}\ \text{(e) passivate}$$

Figure 3.3: The typing rules of λ_μ.

Since passivate and activate always have to be applied consecutively, it is sometimes convenient to combine these rules into one rule.

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash t : \delta \qquad \beta : \delta \in (\Delta, \alpha : \rho)}{\Gamma; \Delta \vdash \mu\alpha : \rho.[\beta]t : \rho}$$

We will use this rule for proofs of certain theorems in which we do not explicitly consider commands.

The typing rules of the previously discussed type systems always had a natural correspondence with some logic. That is, each typing rule corresponds exactly to one logical derivation rule. Here, we have such correspondence with a quite different logic, namely *free deduction* [Par92]. However, we will not present that logic here. Instead, we discuss the relation between λ_μ and minimal classical logic. One direction is easy.

Lemma 3.3.4. *If we have $\Gamma \vdash A$ in minimal classical logic, then $\Gamma; \emptyset \vdash t : A$ for some term t in λ_μ.*

Proof. Peirce's law is typable in λ_μ. That is, for each t of type $(\rho \to \delta) \to \rho$ we can construct a term of type ρ.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x : \rho; \Delta, \alpha : \rho, \beta : \delta \vdash x : \rho}}{\Gamma, x : \rho; \Delta, \alpha : \rho, \beta : \delta \vdash [\alpha]x : \perp\!\!\!\perp}}{\Gamma, x : \rho; \Delta, \alpha : \rho \vdash \mu\beta.[\alpha]x : \delta}}{\Gamma; \Delta, \alpha : \rho \vdash \lambda x.\mu\beta.[\alpha]x : \rho \to \delta} \qquad \Gamma, x : \rho; \Delta, \alpha : \rho, \beta : \delta \vdash t : (\rho \to \delta) \to \rho}{\Gamma; \Delta, \alpha : \rho \vdash t(\lambda x.\mu\beta.[\alpha]x) : \rho}}{\Gamma; \Delta, \alpha : \rho \vdash [\alpha]t(\lambda x.\mu\beta.[\alpha]x) : \perp\!\!\!\perp}}{\Gamma; \Delta \vdash \mu\alpha.[\alpha]t(\lambda x.\mu\beta.[\alpha]x) : \rho}$$

□

One should think of the proof term $\mu\alpha : \rho.[\alpha]t(\lambda x : \rho.\mu\beta : \delta.[\alpha]x)$ as follows. Our goal is ρ, which we label α. Since we have $(\rho \to \delta) \to \rho$ by assumption, it suffices to prove $\rho \to \delta$. Therefore, let us assume ρ, which we label x. Now our goal is δ, which we label β. However, instead of proving goal β we prove an earlier goal, namely α, which simply follows from the assumption x.

The converse is a bit harder, because λ_μ has two environments and minimal classical logic just one. Hence both environments should be mapped onto a single environment. If we have $\Gamma; \Delta \vdash t : \rho$ in λ_μ, then we certainly have $\Gamma, \neg\Delta \vdash \rho$ in classical logic, because activate corresponds with Reduction Ad Absurdum and passivate with negation elimination. However, this does not work for minimal classical logic, because negation cannot be expressed there. So we should transform the environment Δ in a more involved way.

Definition 3.3.5. *Given a term t and a μ-variable β, then a set of simple types*

t_β is defined as follows.

$$\begin{aligned} x_\beta &:= \emptyset \\ (\lambda x.t)_\beta &:= t_\beta \\ (ts)_\beta &:= t_\beta \cup s_\beta \\ (\mu\alpha : \rho.[\gamma]t)_\beta &:= t_\beta \qquad \textit{provided that } \beta \neq \gamma \\ (\mu\alpha : \rho.[\beta]t)_\beta &:= \{\rho\} \cup t_\beta \end{aligned}$$

Moreover, given a term t and an environment of μ-variables Σ, then a set of simple types t_Σ is defined as follows.

$$t_\Sigma := \{\sigma \to \tau \mid \tau \in t_\beta \mid \beta : \sigma \in \Sigma\}$$

Lemma 3.3.6. *If $\Gamma; \Delta \vdash t : \rho$ in λ_μ, then $\Gamma, t_\Delta \vdash \rho$ in minimal classical logic.*

Proof. By induction on the derivation $\Gamma; \Delta \vdash t : \rho$. The only interesting case is activate/passivate, so let $\Gamma; \Delta \vdash \mu\alpha.[\gamma]t : \rho$ with $\Gamma; \Delta, \alpha : \rho \vdash t : \delta$ and $\gamma : \delta \in (\Delta, \alpha : \rho)$. Now we have $\Gamma, t_{(\Delta,\alpha:\rho)} \vdash \delta$ by the induction hypothesis. Furthermore:

$$\begin{aligned} t_{(\Delta,\alpha:\rho)} &= t_\Delta \cup \{\rho \to \tau \mid \tau \in t_\alpha\} \\ &= t_\Delta \cup \{\rho \to \tau_1, \ldots, \rho \to \tau_n\} \end{aligned}$$

for some simple types $\tau_1, \ldots, \tau_n$. Now, by using Peirce's law and $\to$-introduction n times, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, (\mu\alpha.[\gamma]t)_\Delta, \rho \to \tau_1, \ldots, \rho \to \tau_n \vdash \rho}{\Gamma, (\mu\alpha.[\gamma]t)_\Delta, \rho \to \tau_1, \ldots, \rho \to \tau_{n-1} \vdash (\rho \to \tau_n) \to \rho} \quad \ldots \vdash \ldots}{\Gamma, (\mu\alpha.[\gamma]t)_\Delta, \rho \to \tau_1 \vdash \rho}}{\Gamma, (\mu\alpha.[\gamma]t)_\Delta \vdash (\rho \to \tau_1) \to \rho}}{\Gamma, (\mu\alpha.[\gamma]t)_\Delta \vdash \rho}$$

We distinguish the cases $\alpha = \gamma$ and $\alpha \neq \gamma$. In the former case we also have $\delta = \rho$ and $(\mu\alpha.[\gamma]t)_\Delta = t_\Delta$, so we are done. In the latter case we have $(\mu\alpha.[\gamma]t)_\Delta = t_\Delta \cup \{\delta \to \rho\}$, so by thinning and $\to$-elimination we have:

$$\dfrac{\ldots \vdash \delta \to \rho \qquad \dfrac{\Gamma, t_\Delta, \rho \to \tau_1, \ldots, \rho \to \tau_n \vdash \delta}{\Gamma, (\mu\alpha.[\gamma]t)_\Delta, \rho \to \tau_1, \ldots, \rho \to \tau_n \vdash \delta}}{\Gamma, (\mu\alpha.[\gamma]t)_\Delta, \rho \to \tau_1, \ldots, \rho \to \tau_n \vdash \rho}$$

□

Corollary 3.3.7. *If $\Gamma; \emptyset \vdash t : \rho$ in λ_μ, then $\Gamma \vdash \rho$ in minimal classical logic.*

Proof. By Lemma 3.3.6 using the fact that $t_\emptyset = \emptyset$. □

In order to present the reduction rules we need to define an extra notion of substitution: *structural substitution*. Performing structural substitution of a μ-variable β and a call-by-name context (Definition 2.2.7) E for a μ-variable α will recursively replace each command $[\alpha]t$ by $[\beta]E[t']$.

Definition 3.3.8. Structural substitution $t[\alpha := \beta E]$ *of a* μ*-variable* β *and a call-by-name context* E *for a* μ*-variable* α *is defined as follows.*

$$\begin{aligned}
x[\alpha := \beta E] &:= x \\
(\lambda x.r)[\alpha := \beta E] &:= \lambda x.r[\alpha := \beta E] \\
(ts)[\alpha := \beta E] &:= t[\alpha := \beta E]s[\alpha := \beta E] \\
(\mu\alpha.c)[\alpha := \beta E] &:= \mu\alpha.c \\
(\mu\gamma.c)[\alpha := \beta E] &:= \mu\gamma.c[\alpha := \beta E] \qquad \textit{provided that } \gamma \neq \alpha \\
([\alpha]t)[\alpha := \beta E] &:= [\beta]E[t[\alpha := \beta E]] \\
([\gamma]t)[\alpha := \beta E] &:= [\gamma]t[\alpha := \beta E] \qquad \textit{provided that } \gamma \neq \alpha
\end{aligned}$$

Structural substitution is capture avoiding for both λ*- and* μ*-variables.*

Example 3.3.9. *Consider the following examples.*

1. $([\alpha]x\mu\beta.[\alpha]y)[\alpha := \gamma\ (\Box s)] \equiv [\gamma](x\mu\beta.[\gamma]ys)s$
2. $([\alpha]x\mu\beta.[\alpha]y)[\alpha := \beta\ \Box] \equiv [\beta]x\mu\gamma.[\beta]y$
3. $([\alpha]\lambda x.\mu\beta.[\alpha]x)[\alpha := \gamma\ (\Box x)] \equiv [\gamma](\lambda z.\mu\beta.[\gamma]zx)x$

The last two examples illustrate that structural substitution is capture avoiding for both λ*- and* μ*-variables.*

In this thesis we use a notion of structural substitution that is more general than Parigot's original presentation [Par92]. In Parigot's original presentation we have $t[\beta := \alpha]$, which renames each μ-variable β into α, and $t[\alpha := s]$, which replaces each command $[\alpha]t$ by $[\alpha]t's$. Of course, Parigot's notions are just instances of our definition, namely, the former corresponds to $t[\beta := \alpha\ \Box]$ and the latter to $t[\alpha := \alpha\ (\Box s)]$. Although Parigot's presentation suffices for definition of the reduction rules, our presentation turns out to be more convenient for extensions of λ_μ (Section 4.2) and for proving properties like confluence (Section 4.3) and strong normalization (Section 4.4).

Definition 3.3.10. *Reduction* $t \to t'$ *on* λ_μ*-terms* t *and* t' *is defined as the compatible closure of the following rules.*

$$\begin{aligned}
(\lambda x.t)r &\to_\beta t[x := r] \\
(\mu\alpha.c)s &\to_{\mu R} \mu\alpha.c[\alpha := \alpha\ (\Box s)] \\
\mu\alpha.[\alpha]t &\to_{\mu\eta} t \qquad \textit{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c &\to_{\mu i} c[\beta := \alpha\ \Box]
\end{aligned}$$

We introduce a similar notation as for λ_Δ.

Notation 3.3.11. $\Theta c := \mu\gamma : \rho.c$ *provided that* $\gamma \notin \mathrm{FCV}(c)$.

From a computational point of view one should think of $\mu\alpha.[\beta]t$ in the same way as of $\Delta x.yt$. It behaves like a combined catch and throw clause: it catches exceptions labeled α in t and finally throws the results of t to $\mu\beta.c$.

Definition 3.3.12. *The terms* $\texttt{catch}\ \alpha\ t$ *and* $\texttt{throw}\ \beta\ s$ *are defined as follows.*

$$\texttt{catch}\ \alpha\ t := \mu\alpha.[\alpha]t$$
$$\texttt{throw}\ \beta\ s := \Theta[\beta]s$$

Lemma 3.3.13. *We have the following reductions for* `catch` *and* `throw`*.*

1. $E[\texttt{throw}\ \alpha\ t] \twoheadrightarrow \texttt{throw}\ \alpha\ t$
2. $\texttt{catch}\ \alpha\ (\texttt{throw}\ \alpha\ t) \twoheadrightarrow \texttt{catch}\ \alpha\ t$
3. $\texttt{catch}\ \alpha\ t \twoheadrightarrow t$ *provided that* $\alpha \notin \mathrm{FCV}(t)$
4. $\texttt{throw}\ \beta\ (\texttt{throw}\ \alpha\ s) \to \texttt{throw}\ \alpha\ s$

Proof. These reductions follow directly from the reduction rules of λ_μ, except for the first one, where an induction on the structure of E is needed. □

Just like the ordinary λ-calculus, the λ_μ-calculus satisfies the main meta theoretical properties. We treat these properties now.

Lemma 3.3.14. λ_μ *is confluent.*

Parigot's original proof sketch [Par92], which is based on the notion of parallel reduction by Tait and Martin-Löf, is wrong (this was first noticed by Fujita in [Fuj97]). As observed in [Fuj97, BHF01], the usual notion of parallel reduction does not extend well to λ_μ: it only allows to prove weak confluence. But since λ_μ is strongly normalizing (Lemma 3.3.17) we have confluence for well-typed terms by Newman's lemma. But confluence is a property that also holds for untyped terms, so this result is not quite satisfactory. Confluence for untyped λ_μ-terms can be proven by analogy to the proof in Section 4.3. For now, we postpone a discussion of the niceties of this proof.

Lemma 3.3.15. *Typing is preserved under structural substitution. That is, if:*

1. $\Gamma; \Delta, \alpha : \delta \vdash t : \rho$*, and,*
2. $\Gamma'; \Delta' \vdash E[r] : \gamma$ *for all environments* $\Gamma' \supseteq \Gamma$*,* $\Delta' \supseteq \Delta$ *and terms* r *such that* $\Gamma'; \Delta' \vdash r : \delta$*,*

then $\Gamma; \Delta, \beta : \gamma \vdash t[\alpha := \beta E] : \rho$*.*

Proof. By induction on the derivation $\Gamma; \Delta, \alpha : \delta \vdash t : \rho$. The only interesting case is passivate, so let $\Gamma; \Delta, \alpha : \delta \vdash [\alpha]t : \perp\!\!\!\perp$ with $\Gamma; \Delta, \alpha : \delta \vdash t : \delta$. Now we have $\Gamma; \Delta, \beta : \gamma \vdash t[\alpha := \beta E] : \delta$ by the induction hypothesis. Thus by assumption $\Gamma; \Delta, \beta : \gamma \vdash E[t[\alpha := \beta E]] : \gamma$, so $\Gamma; \Delta, \beta : \gamma \vdash [\beta]E[t[\alpha := \beta E]] : \perp\!\!\!\perp$. □

Lemma 3.3.16. λ_μ *satisfies subject reduction.*

Proof. First we have to prove a similar substitution lemma as we have proven for $\lambda\!\!\to$ (Lemma 2.3.7). Then we have to prove that all reduction rules preserve typing. We treat some interesting reduction rules.

1. The $\rightarrow_{\mu R}$-rule:

$$\frac{\dfrac{c : \perp\!\!\!\perp}{\mu\alpha.c : \rho \rightarrow \delta} \qquad s : \rho}{(\mu\alpha.c)s : \delta} \quad \rightarrow_{\mu R} \quad \frac{c[\alpha := \alpha\ (\Box s)] : \perp\!\!\!\perp}{\mu\alpha.c[\alpha := \alpha\ (\Box s)] : \delta}$$

   It remains to prove that $\Gamma; \Delta, \alpha : \delta \vdash c[\alpha := \alpha\ (\Box s)] : \perp\!\!\!\perp$. We proceed by applying Lemma 3.3.15, so given contexts $\Gamma' \supseteq \Gamma$, $\Delta' \supseteq \Delta$ and a derivation $\Gamma'; \Delta' \vdash r : \rho \rightarrow \delta$ then we have to prove that $\Gamma'; \Delta' \vdash rs : \delta$. The required result is shown below.

$$\frac{\Gamma'; \Delta' \vdash r : \rho \rightarrow \delta \qquad \dfrac{\Gamma; \Delta \vdash s : \rho}{\Gamma'; \Delta' \vdash s : \rho}}{\Gamma'; \Delta' \vdash rs : \delta}$$

2. The $\rightarrow_{\mu\eta}$-rule:

$$\frac{\dfrac{t : \rho}{[\alpha]t : \perp\!\!\!\perp}}{\mu\alpha.[\alpha]t : \rho} \quad \rightarrow_{\mu\eta} \quad t : \rho$$

3. The $\rightarrow_{\mu i}$-rule:

$$\frac{\dfrac{c : \perp\!\!\!\perp}{\mu\beta.c : \rho}}{[\alpha]\mu\beta.c : \perp\!\!\!\perp} \quad \rightarrow_{\mu i} \quad c[\beta := \alpha\ \Box] : \perp\!\!\!\perp$$

   Here we have $c[\beta := \alpha\ \Box] : \perp\!\!\!\perp$ by Lemma 3.3.15. □

Lemma 3.3.17. *λ_μ is strongly normalizing.*

Proof. This is proven in [Par97]. □

In Section 2.3 we have shown that various data types can be encoded in the $\lambda{\rightarrow}$-calculus. Because $\lambda{\rightarrow}$ is a subsystem of λ_μ we can of course reuse that methodology. Unfortunately, as noticed in [Par92], unique representation of data types in λ_μ fails. For $\lambda{\rightarrow}$-calculus, we had a one-to-one correspondence between closed normal forms of type N_γ and natural numbers (Corollary 2.3.14). But this property is not preserved as the following example illustrates.

$$\lambda f : \gamma \rightarrow \gamma.\lambda x : \gamma.\mu\alpha : \gamma\,.\,[\alpha]f(\Theta[\alpha]x)$$

Also, in $\lambda{\rightarrow}$, closed normal forms of type $\rho \rightarrow \delta$ were of the shape $t \equiv \lambda x.r$, but this result is not preserved either. For example, consider:

$$s \equiv \mu\alpha : \gamma \rightarrow \gamma\,.\,[\alpha]\lambda x : \gamma\,.\,\Theta[\alpha]\lambda y : \gamma\,.\,y$$

Here, s computes just the identity. We see this by reducing its η-expansion.

$$\begin{aligned} \lambda z.sz &\equiv \lambda z.(\mu\alpha.[\alpha]\lambda x.\Theta[\alpha]\lambda y.y)z \\ &\rightarrow \lambda z.\mu\alpha.[\alpha](\lambda x.\Theta[\alpha](\lambda y.y)z)z \\ &\twoheadrightarrow \lambda z.\mu\alpha.[\alpha]\Theta[\alpha]z \\ &\twoheadrightarrow \lambda z.\mu\alpha.[\alpha]z \\ &\twoheadrightarrow \lambda z.z \end{aligned}$$

This example moreover indicates that adding the η-rule[1] to the λ_μ-calculus

[1] $\lambda z.tz \rightarrow t$ provided that $z \notin \mathrm{FV}(t)$

results in failure of the confluence property.

3.4 The second-order λ_μ-calculus

In this section we present the second-order λ_μ-calculus (henceforth $\lambda_\mu^{\mathbf{2}}$) by Parigot [Par92, Par97]. The $\lambda_\mu^{\mathbf{2}}$-calculus is basically a combination of λ_μ and $\lambda^{\mathbf{2}}$. Moreover, it extends the Curry-Howard correspondence to second-order classical propositional logic.

Definition 3.4.1. *The* terms *and* commands *of $\lambda_\mu^{\mathbf{2}}$ are mutually inductively defined over an infinite set of* λ-variables $(x, y, \ldots)$ *and* μ-variables $(\alpha, \beta, \ldots)$ *as follows.*

$$\begin{aligned} t, r, s &::= x \mid \lambda x : \rho.r \mid ts \mid \lambda\gamma.t \mid t\rho \mid \mu\alpha : \rho.c \\ c, d &::= [\alpha]t \end{aligned}$$

Here, ρ ranges over second-order types (Definition 2.5.1).

Remark 3.4.2. *Note that we may the names α and β for both type variables and μ-variables. However, it will always be clear whether we mean a type variable or μ-variable.*

As usual, we let $\mathrm{FV}(t)$, $\mathrm{FCV}(t)$ and $\mathrm{FTV}(t)$ denote the set of free λ-variables, free μ-variables and free type variables of a term t, respectively. Moreover, the operation of capture avoiding substitution $t[x := r]$ of r for x in t and capture avoiding substitution $t[\alpha := \rho]$ of ρ for α in t generalize to $\lambda_\mu^{\mathbf{2}}$-terms in the obvious way.

Definition 3.4.3. *A $\lambda_\mu^{\mathbf{2}}$-typing judgment $\Gamma; \Delta \vdash t : \rho$ denotes that a term t has type ρ in an environment of λ-variables Γ and an environment of μ-variables Δ. A typing judgment $\Gamma; \Delta \vdash c : \perp\!\!\!\perp$ denotes that a command c is typable in an environment of λ-variables Γ and an environment of μ-variables Δ. The derivation rules for such judgments are mutually recursively defined and shown in Figure 3.4.*

$$\frac{x : \rho \in \Gamma}{\Gamma; \Delta \vdash x : \rho}$$
(a) axiom

$$\frac{\Gamma, x : \rho; \Delta \vdash t : \delta}{\Gamma; \Delta \vdash \lambda x : \rho.t : \rho \to \delta}$$
(b) lambda

$$\frac{\Gamma; \Delta \vdash t : \rho \to \delta \qquad \Gamma; \Delta \vdash s : \rho}{\Gamma; \Delta \vdash ts : \delta}$$
(c) app

$$\frac{\Gamma; \Delta \vdash t : \rho}{\Gamma; \Delta \vdash \lambda\gamma.t : \forall\gamma.\rho}\ \gamma \notin \mathrm{FTV}(\Gamma)$$
(d) $\forall_i$

$$\frac{\Gamma; \Delta \vdash t : \forall\gamma.\rho}{\Gamma; \Delta \vdash t\delta : \rho[\gamma := \delta]}$$
(e) $\forall_e$

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash c : \perp\!\!\!\perp}{\Gamma; \Delta \vdash \mu\alpha : \rho.c : \rho}$$
(f) activate

$$\frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash [\alpha]t : \perp\!\!\!\perp}$$
(g) passivate

Figure 3.4: The typing rules of $\lambda_\mu^{\mathbf{2}}$.

In order to extend the Curry-Howard correspondence to second-order classical logic we have to show that $\lambda_\mu^{\mathbf{2}}$-judgments correspond to judgments in second-order classical logic. As we have seen in the Section 3.3, a similar result for the simply-typed λ_μ-calculus took quite some work. In a second-order system it becomes easier because we can define a connective false. So, if we have $\Gamma; \Delta \vdash t : \rho$ in $\lambda_\mu^{\mathbf{2}}$, then we have $\Gamma, \neg\Delta \vdash \rho$ in second-order classical logic, because activate corresponds with Reduction Ad Absurdum and passivate with negation elimination. For the converse we have to show that Peirce's law is inhibited in $\lambda_\mu^{\mathbf{2}}$, this result is similar to Lemma 3.3.4.

Theorem 3.4.4. *We have $\Gamma, \neg\Delta \vdash A$ in second-order classical propositional logic iff $\Gamma; \Delta \vdash t : A$ for some term t in $\lambda_\mu^{\mathbf{2}}$.*

For this extension we already see that our definition of structural substitution pays off: we just have to extend call-by-name contexts with a new constructor.

Definition 3.4.5. *A* $\lambda_\mu^{\mathbf{2}}$-context *is defined as follows.*

$$E ::= \Box \mid Et \mid E\rho$$

Now we generalize the notion of substitution $E[s]$ of s for the hole $\Box$ in E and the notion of structural substitution $t[\alpha := \beta E]$ of βE for α in t in a straightforward way.

Definition 3.4.6. *Reduction $t \to t'$ on $\lambda_\mu^{\mathbf{2}}$-terms t and t' is defined as the compatible closure of the rules displayed in Figure 3.5. As usual, $\twoheadrightarrow^+$ denotes the transitive closure, $\twoheadrightarrow$ denotes the reflexive/transitive closure and $=$ denotes the reflexive/symmetric/transitive closure.*

$$\begin{array}{lll}
(\lambda x.t)r & \to_\beta & t[x := r] \\
(\lambda\gamma.t)\rho & \to_{\beta\forall} & t[\gamma := \rho] \\
(\mu\alpha.c)s & \to_{\mu R} & \mu\alpha.c[\alpha := \alpha\ (\Box s)] \\
(\mu\alpha.c)\rho & \to_{\mu\forall} & \mu\alpha.c[\alpha := \alpha\ (\Box\rho)] \\
\mu\alpha.[\alpha]t & \to_{\mu\eta} & t \qquad \text{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c & \to_{\mu i} & c[\beta := \alpha\ \Box]
\end{array}$$

Figure 3.5: The reduction rules of $\lambda_\mu^{\mathbf{2}}$.

Just like the λ_μ- and $\lambda^{\mathbf{2}}$-calculus, the $\lambda_\mu^{\mathbf{2}}$-calculus is confluent, satisfies satisfies subject reduction and is strongly normalizing [Par92, Par97]. However, we will not go into further details here.

Just as in λ_μ, we do not have unique representation of data types. This property fails because of the same reasons as we have shown in Section 3.3. However, in [Par93], this defect is solved by means of the *output operator* Φ, which extracts the actual numeral from a $\lambda_\mu^{\mathbf{2}}$-term.

Definition 3.4.7. *The* output operator Φ *is defined as follows.*

$$\Phi := \lambda n.n\,((\mathsf{N} \to \mathsf{N}) \to \mathsf{N})\,\hat{\mathsf{S}}\,\hat{0}\,\mathsf{I}$$

Here, $\hat{0} := \lambda k.kc_0$ *and* $\hat{\mathsf{S}} := \lambda kh.k\,(\lambda l.h\,(\mathsf{S}l))$.

Fact 3.4.8. *The output operator* Φ *is well-typed. That is* $\hat{0} : (\mathsf{N} \to \mathsf{N}) \to \mathsf{N}$, $\hat{\mathsf{S}} : ((\mathsf{N} \to \mathsf{N}) \to \mathsf{N}) \to (\mathsf{N} \to \mathsf{N}) \to \mathsf{N}$ *and* $\Phi : \mathsf{N} \to \mathsf{N}$.

Lemma 3.4.9. *Given a closed term* $t : \mathsf{N}$*, then* $\Phi\, t \twoheadrightarrow c_m$ *for some* $m \in \mathbb{N}$.

Proof. Proven in [Par93]. □

To illustrate the behavior of the output operator Φ, we will apply it to the term $t \equiv \lambda\gamma.\lambda fx\,.\,\mu\alpha.[\alpha]f(\Theta[\alpha]fx)$.

$$\begin{aligned}
\Phi\, t &\twoheadrightarrow (\lambda fx\,.\,\mu\alpha.[\alpha]f(\Theta[\alpha]fx))\,\hat{\mathsf{S}}\,\hat{0}\,\mathsf{I} \\
&\twoheadrightarrow (\mu\alpha.[\alpha]\hat{\mathsf{S}}\,(\Theta[\alpha]\hat{\mathsf{S}}\,\hat{0}))\,\mathsf{I} \\
&\twoheadrightarrow \mu\alpha.[\alpha]\hat{\mathsf{S}}\,(\Theta[\alpha]\hat{\mathsf{S}}\,\hat{0}\,\mathsf{I})\,\mathsf{I} \\
&\twoheadrightarrow \mu\alpha.[\alpha](\Theta[\alpha]\hat{\mathsf{S}}\,\hat{0}\,\mathsf{I})\,(\lambda\mathsf{I}.\mathsf{I}\,(\mathsf{SI})) \\
&\twoheadrightarrow \mu\alpha.[\alpha]\Theta[\alpha]\hat{\mathsf{S}}\,\hat{0}\,\mathsf{I} \\
&\twoheadrightarrow \hat{\mathsf{S}}\,\hat{0}\,\mathsf{I} \\
&\twoheadrightarrow \mathsf{I}\,(\mathsf{S}c_0) \\
&\twoheadrightarrow \mathsf{S}c_0 \\
&\twoheadrightarrow c_1
\end{aligned}$$

If we write t as $t \equiv \lambda\gamma.\lambda fx\,.\,\texttt{catch}\ \alpha\ f(\texttt{throw}\ \alpha\ (fx))$, then this result is exactly what one would expect, because the `throw` α (fx) should jump to the `catch`. The idea of such a output operator is closely related to continuation passing style, a notion which will be explained in the next section.

3.5 Continuation passing style

Continuation passing style (CPS) is a style of programming that is suitable to make control and evaluation explicit. Therefore, it is particular useful to simulate control operators in a system without. Before we present such a simulation of $\lambda_\mu^{\mathbf{2}}$ in $\lambda^{\mathbf{2}}$ we explain the basic idea of CPS.

In *direct style*, the most commonly used style of programming, a function just *returns* its result. For example, let us consider the following program, which takes a number n and computes the Fibonacci number $F(n)$, in direct style.

```
let rec fib n = match n with
  | 0    -> 0
  | 1    -> 1
  | SSm  -> fib (Sm) + fib m
```

In order to compute the Fibonacci number $F(\texttt{SS}m)$, it has to make recursive calls to both `fib` $(\texttt{S}m)$ and `fib` m, hence this program is not primitive recursive.

In continuation passing style each function is parametrized by a function, the so called *continuation*, which is invoked with the function's result. Now we will use CPS to write a primitive recursive program that computes the Fibonacci numbers[2].

[2] Alternatively, one could let `fib2` n yield a pair of $F(n)$ and $F(n+1)$.

```
let fib n = fib2 n (λk l.k)

let rec fib2 n f = match n with
  | 0   -> f 0 1
  | Sm  -> fib2 m (λk l.f l (k+l))
```

Now, instead of returning its result, evaluation of `fib2` $n\,f$ invokes the function f with the Fibonacci numbers $F(n)$ and $F(n+1)$. To use the function `fib2` in direct style, we have to apply the so called *top-level continuation* $(\lambda k\,l\,.\,k)$, which picks the Fibonacci number $F(n)$. Let us illustrate the behavior of this program by computing the second Fibonacci number.

$$\begin{aligned}
\texttt{fib}\ 2 &\equiv \texttt{fib2}\ 2\ (\lambda k_1\,l_1\,.\,k_1)\\
&\to \texttt{fib2}\ 1\ (\lambda k_2\,l_2\,.\,(\lambda k_1\,l_1\,.\,k_1)\ l_2\ (k_2+l_2))\\
&\to \texttt{fib2}\ 0\ (\lambda k_3\,l_3\,.\,(\lambda k_2\,l_2\,.\,(\lambda k_1\,l_1\,.\,k_1)\ l_2\ (k_2+l_2))\ l_3\ (k_3+l_3))\\
&\to (\lambda k_3\,l_3\,.\,(\lambda k_2\,l_2\,.\,(\lambda k_1\,l_1\,.\,k_1)\ l_2\ (k_2+l_2))\ l_3\ (k_3+l_3))\ 0\ 1\\
&\to (\lambda k_2\,l_2\,.\,(\lambda k_1\,l_1\,.\,k_1)\ l_2\ (k_2+l_2))\ 1\ (0+1)\\
&\to (\lambda k_1\,l_1\,.\,k_1)\ (0+1)\ (1+(0+1))\\
&\to 0+1\\
&\to 1
\end{aligned}$$

Notice that the continuation is behaving as a stack that contains the remainder of the required computation.

More interestingly, CPS can be used to simulate control and evaluation order. This idea originates from Plotkin who used CPS to simulate the λ_v-calculus in the ordinary λ-calculus [Plo75]. Plotkin's *CPS-translation* parametrizes each subterm t with a function that, when invoked with the result of t, returns the overall result.

First we list some desired theoretical properties of CPS-translations. Therefore, let t° denote some CPS-translation of an arbitrary term t in some source system $\mathfrak{S}$ (e.g. $\lambda_\mu^{\mathbf{2}}$) into some target system $\mathfrak{T}$ (e.g. $\lambda^{\mathbf{2}}$) and let ρ° denote a corresponding translation for types.

Soundness. If $t_1 = t_2$ in $\mathfrak{S}$, then ${t_1}^\circ = {t_2}^\circ$ in $\mathfrak{T}$.

Completeness. If ${t_1}^\circ = {t_2}^\circ$ in $\mathfrak{T}$, then $t_1 = t_2$ in $\mathfrak{S}$.

Preservation of reduction. If $t_1 \to t_2$ in $\mathfrak{S}$, then ${t_1}^\circ \twoheadrightarrow^+ {t_2}^\circ$ in $\mathfrak{T}$.

Preservation of typing. If $\Gamma \vdash t : \rho$ in $\mathfrak{S}$, then $\Gamma^\circ \vdash t^\circ : \rho^\circ$ in $\mathfrak{T}$.

Now we will relate the previously described theoretical properties to actual applications of CPS-translations.

1. Type preserving CPS-translations can be used to give an interpretation of classical logic in intuitionistic logic. We will give an interpretation of second-order classical propositional logic in minimal second-order propositional logic in this section.

2. In order to show that $\mathfrak{S}$ has certain expressive power, it is sufficient to embed $\mathfrak{S}$ into $\mathfrak{T}$, where $\mathfrak{T}$ is a system whose expressive power is already known. This can be achieved by means of a sound CPS-translation which is also adequate with respect to encoding of data types. An example of such embedding will be given in Section 4.5.

3. Instead of proving a certain property of terms in system $\mathfrak{S}$, one could prove that property, using a CPS-translation, in system $\mathfrak{T}$. Here it is essential that the CPS-translation is both sound and complete.

4. Strong normalization proofs are usually quite a lot of work. However, a reduction preserving CPS-translation of $\mathfrak{S}$ into $\mathfrak{T}$ can be used in case $\mathfrak{T}$ is known to be strongly normalizing. Here we reason by contradiction. We assume that an infinite reduction sequence in $\mathfrak{S}$ exists and translate it into an infinite reduction sequence in $\mathfrak{T}$. However, $\mathfrak{T}$ is strongly normalization, so we obtain a contradiction. Furthermore, the system $\mathfrak{T}$ is usually only strongly normalizing for well-typed terms, so we need a CPS-translation that is type preserving as well.

In the remainder of this section we will present a sound and type preserving CPS-translation of $\lambda^{\mathbf{2}}_{\mu}$ into $\lambda^{\mathbf{2}}$. First we present the translation of types, which is also known as the *Kolmogorov double negation translation.*

Definition 3.5.1. *Given a type τ, then let $\neg\rho$ denote $\rho \to \tau$. Now given a type ρ, then the* negative translation *ρ° of ρ is mutually inductively defined with $\rho^\bullet$ as follows.*

$$\begin{aligned} \rho^\circ &:= \neg\neg\rho^\bullet \\ \alpha^\bullet &:= \alpha \\ (\rho \to \delta)^\bullet &:= \rho^\circ \to \delta^\circ \\ (\forall\alpha.\rho)^\bullet &:= \forall\alpha.\rho^\circ \end{aligned}$$

Lemma 3.5.2. *Given types ρ and δ, then $\rho^\bullet[\alpha := \delta^\bullet] = (\rho[\alpha := \delta])^\bullet$.*

Proof. Straightforward by induction on the structure of ρ. □

In order to make sure that this translation makes sense, we prove that σ is logically equivalent to σ° in second-order classical propositional logic.

Lemma 3.5.3. *For the negative translation set $\tau = \bot$. Now we have $\Gamma \vdash \sigma \to \sigma^\circ$ and $\Gamma \vdash \sigma^\circ \to \sigma$ in second-order classical propositional logic.*

Proof. Simultaneously by induction on the structure of σ.

1. Suppose that $\sigma = \alpha$. The first property is immediate. The second property follows from double negation elimination.

2. Suppose that $\sigma = \rho \to \delta$. Now, by the induction hypothesis, we have $\Gamma \vdash \rho \to \rho^\circ$, $\Gamma \vdash \rho^\circ \to \rho$, $\Gamma \vdash \delta \to \delta^\circ$ and $\Gamma \vdash \delta^\circ \to \delta$ for each environment Γ. Hence $\Gamma \vdash (\rho \to \delta) \to (\rho \to \delta)^\circ$ as shown below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\rho^\circ \to \rho}\quad\dfrac{}{[\rho^\circ]}}{\rho}}{[\rho \to \delta]}}{\delta}}{\delta \to \delta^\circ}}{\delta^\circ}\quad\dfrac{}{\rho^\circ \to \delta^\circ}}{[\neg(\rho \to \delta)^\bullet]}}{\bot}}{(\rho \to \delta)^\circ}}{(\rho \to \delta) \to (\rho \to \delta)^\circ}$$

Moreover, we have $\Gamma \vdash (\rho \to \delta)^\circ \to \rho \to \delta$ as shown below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\rho \to \rho^\circ}\quad\dfrac{}{[\rho]}}{\rho^\circ}}{[\rho^\circ \to \delta^\circ]}}{\delta^\circ}\quad\dfrac{}{[\neg\delta^\bullet]}}{\bot}}{\neg(\rho^\circ \to \delta^\circ)}}{[(\rho \to \delta)^\circ]}}{\bot}}{\delta^\circ}}{\delta^\circ \to \delta}}{\delta}}{(\rho \to \delta)^\circ \to \rho \to \delta}$$

3. Suppose that $\sigma = \forall\alpha.\rho$. This case is similar to the preceding case. □

Now we define the actual CPS-translation of $\lambda_\mu^{\mathbf{2}}$ into $\lambda^{\mathbf{2}}$. This translation is pretty much straightforward keeping in mind that the CPS-translation of each subterm t takes a continuation that has to be invoked with the value of t.

Definition 3.5.4. *Given a $\lambda_\mu^{\mathbf{2}}$-term t, then the* CPS-translation t° *of t into $\lambda^{\mathbf{2}}$ is inductively defined as follows.*

$$\begin{aligned}
x^\circ &:= \lambda k.xk \\
(\lambda x.t)^\circ &:= \lambda k.k(\lambda x.t^\circ) \\
(tr)^\circ &:= \lambda k.t^\circ(\lambda l.lr^\circ k) \\
(\lambda\gamma.t)^\circ &:= \lambda k.k(\lambda\gamma.t^\circ) \\
(t\rho)^\circ &:= \lambda k.t^\circ(\lambda l.l\rho^\bullet k) \\
(\mu\alpha.c)^\circ &:= \lambda k_\alpha.c^\circ \\
([\alpha]t)^\circ &:= t^\circ k_\alpha
\end{aligned}$$

Here, k_α is a fresh λ-variable for each μ-variable α.

Let us take a look at the application case. Here we take a continuation k, which has to be invoked with the value of tr. In order to obtain such a value, the term t has to be reduced to a value. To do so, we have to supply it with a continuation that will be invoked with the value of t. We can construct such a continuation by taking the value l of t. The variable l, which gets bound with the value of t, again needs a continuation. The natural choice is r°, but now lr° expects a continuation that should be invoked with the value of tr. Of course, that should be k.

If we would consider a call-by-value system instead, then the translation of the application becomes $(tr)^\circ := \lambda k.r^\circ(\lambda m.t^\circ(\lambda l.lmk)$. This translation ensures that r is reduced to a value before further reduction may happen.

Lemma 3.5.5. *The translation from $\lambda_\mu^{\mathbf{2}}$ into $\lambda^{\mathbf{2}}$ preserves typing. That is:*

$$\Gamma; \Delta \vdash t : \rho \text{ in } \lambda_\mu^{\mathbf{2}} \qquad \Longrightarrow \qquad \Gamma^\circ, \Delta^\circ \vdash t^\circ : \rho^\circ \text{ in } \lambda^{\mathbf{2}}$$

where $\Gamma^\circ = \{x : \rho^\circ \mid x : \rho \in \Gamma\}$ *and* $\Delta^\circ = \{k_\alpha : \neg\rho^\bullet \mid \alpha : \rho \in \Delta\}$.

Proof. We prove that we have $\Gamma; \Delta \vdash t : \rho$ and $\Gamma; \Delta \vdash c : \perp\!\!\!\perp$ by mutual induction on the derivations $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \rho^\circ$ and $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \perp$, respectively.

(var) Let $\Gamma; \Delta \vdash x : \rho$ such that $x : \rho \in \Gamma$. Now we have $x : \rho^\circ \in \Gamma^\circ$ and so $\Gamma^\circ, \Delta^\circ \vdash x^\circ : \rho^\circ$ as shown below.

$$\dfrac{\dfrac{x : \rho^\circ \qquad k : \neg\rho^\bullet}{xk : \perp}}{\lambda k.xk : \rho^\circ}$$

(λ) Let $\Gamma; \Delta \vdash \lambda x.t : \rho \to \delta$ with $\Gamma, x : \rho; \Delta \vdash t : \delta$. Now $\Gamma^\circ, x : \rho^\circ, \Delta^\circ \vdash t^\circ : \delta^\circ$ by the induction hypothesis. So $\Gamma^\circ, \Delta^\circ \vdash (\lambda x.t)^\circ : (\rho \to \delta)^\circ$ as shown below.

$$\dfrac{\dfrac{k : \neg(\rho \to \delta)^\bullet \qquad \dfrac{t^\circ : \delta^\circ}{\lambda x.t^\circ : \rho^\circ \to \delta^\circ}}{k(\lambda x.t^\circ) : \perp}}{\lambda k.k(\lambda x.t^\circ) : (\rho \to \delta)^\circ}$$

(app) Let $\Gamma; \Delta \vdash tr : \delta$ with $\Gamma; \Delta \vdash t : \rho \to \delta$ and $\Gamma; \Delta \vdash s : \rho$. Now we have $\Gamma^\circ, \Delta^\circ \vdash t^\circ : (\rho \to \delta)^\circ$ and $\Gamma^\circ, \Delta^\circ \vdash r^\circ : \rho^\circ$ by the induction hypothesis. So $\Gamma^\circ, \Delta^\circ \vdash tr^\circ : \delta^\circ$ as shown below.

$$\dfrac{\dfrac{t^\circ : (\rho \to \delta)^\circ \qquad \dfrac{\dfrac{\dfrac{l : \rho^\circ \to \delta^\circ \qquad r^\circ : \rho^\circ}{lr^\circ : \delta^\circ} \qquad k : \neg\delta^\bullet}{lr^\circ k : \perp}}{\lambda l.lr^\circ k : \neg(\rho^\circ \to \delta^\circ)}}{t^\circ(\lambda l.lr^\circ k) : \perp}}{\lambda k.t^\circ(\lambda l.lr^\circ k) : \delta^\circ}$$

($\forall_i$) Let $\Gamma; \Delta \vdash \lambda\gamma.t : \forall\gamma.\rho$ with $\Gamma; \Delta \vdash t : \rho$. Now $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \rho^\circ$ by the induction hypothesis. So $\Gamma^\circ, \Delta^\circ \vdash (\lambda\gamma.t)^\circ : (\forall\gamma.\rho)^\circ$ as shown below.

$$\dfrac{\dfrac{k : \neg(\forall\gamma.\rho)^\bullet \qquad \dfrac{t^\circ : \rho^\circ}{\lambda\gamma.t^\circ : \forall\gamma.\rho^\circ}}{k(\lambda\gamma.t^\circ) : \perp}}{\lambda k.k(\lambda\gamma.t^\circ) : (\forall\gamma.\rho)^\circ}$$

($\forall_e$) Let $\Gamma; \Delta \vdash t\delta : \rho[\gamma := \delta]$ with $\Gamma; \Delta \vdash t : \forall\gamma.\rho$. Now $\Gamma^\circ, \Delta^\circ \vdash t^\circ : (\forall\gamma.\rho)^\circ$ by the induction hypothesis. So $\Gamma^\circ, \Delta^\circ \vdash t\delta^\circ : (\rho[\gamma := \delta])^\circ$ as shown below.

$$\dfrac{\dfrac{t^\circ : (\forall\gamma.\rho)^\circ \qquad \dfrac{\dfrac{\dfrac{l : \forall\gamma.\rho^\circ}{l\delta^\bullet : \rho^\circ[\gamma := \delta^\bullet]} \qquad k : (\neg\rho^\bullet)[\gamma := \delta^\bullet]}{l\delta^\bullet k : \bot}}{\lambda l.l\delta^\bullet k : \neg(\forall\gamma.\rho^\circ)}}{t^\circ(\lambda l.l\delta^\bullet k) : \bot}}{\lambda k.t^\circ(\lambda l.l\delta^\bullet k) : (\rho[\gamma := \delta])^\circ}$$

Here we have $(\neg\rho^\bullet)[\gamma := \delta^\bullet] = \neg(\rho[\gamma := \delta])^\bullet$ by Lemma 3.5.2.

(act) Let $\Gamma; \Delta \vdash \mu\alpha.c : \rho$ with $\Gamma; \Delta, \alpha : \rho \vdash c : \perp\!\!\!\perp$. By the induction hypothesis we have $\Gamma^\circ, \Delta^\circ, k_\alpha : \neg\rho^\bullet \vdash c^\circ : \bot$. So $\Gamma^\circ, \Delta^\circ \vdash (\mu\alpha.c)^\circ : \rho^\circ$ as shown below.

$$\dfrac{c^\circ : \bot}{\lambda k_\alpha.c^\circ : \rho^\circ}$$

(pas) Let $\Gamma; \Delta \vdash [\alpha].t : \perp\!\!\!\perp$ with $\alpha : \delta \in \Delta$. Now we have $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \rho^\circ$ by the induction hypothesis. Furthermore we have $k_\alpha : \neg\rho^\bullet \in \Delta^\circ$ and so $\Gamma^\circ, \Delta^\circ \vdash ([\alpha].t)^\circ : \bot$ as shown below.

$$\dfrac{t^\circ : \rho^\circ \qquad k_\alpha : \neg\rho^\bullet}{t^\circ k_\alpha : \bot}$$

□

To prove that our CPS-translation is sound we are required to prove some auxiliary substitution lemmas first.

Lemma 3.5.6. *Given a $\lambda_\mu^{\mathbf{T}}$-term t, then $\lambda k.t^\circ k \to t^\circ$.*

Proof. This follows immediately from the Definition 3.5.4, because the CPS-translation t° of t is of the shape $\lambda l.t'$, so $\lambda k.(\lambda l.t')k \twoheadrightarrow \lambda k.t'[l := k] \equiv t^\circ$. □

Lemma 3.5.7. *Given $\lambda_\mu^{\mathbf{T}}$-terms t and r, then $t^\circ[x := r^\circ] \twoheadrightarrow (t[x := r])^\circ$.*

Proof. By induction on the structure of t. The only interesting case is $t \equiv x$.

$$\begin{aligned} x^\circ[x := r^\circ] &\equiv (\lambda k.xk)[x := r^\circ] \\ &\equiv \lambda k.r^\circ k \\ &\twoheadrightarrow r^\circ \qquad \text{(a)} \\ &\equiv (x[x := r])^\circ \end{aligned}$$

Here, step (a) holds by Lemma 3.5.6. □

Lemma 3.5.8. *Given a $\lambda_\mu^{\mathbf{T}}$-term t, then:*

1. $(t[\alpha := \beta\ \Box])^\circ \equiv t^\circ[k_\alpha := k_\beta]$
2. $(t[\alpha := \beta\ (\Box s)])^\circ \twoheadrightarrow t^\circ[k_\alpha := \lambda l.ls^\circ k_\beta]$
3. $(t[\alpha := \beta\ (\Box\rho)])^\circ \twoheadrightarrow t^\circ[k_\alpha := \lambda l.l\rho^\bullet k_\beta]$

Proof. All properties are proven by induction on the structure of t. The only interesting case is $c \equiv [\alpha]t$.

1. Property 1: here we have $(t[\alpha := \beta\ \Box])^\circ \equiv t^\circ[k_\alpha := k_\beta]$ by the induction hypothesis, so:

$$\begin{aligned}(([\alpha]t)[\alpha := \beta\ \Box])^\circ &\equiv t[\alpha := \beta\ \Box]^\circ k_\beta \\ &\equiv t^\circ[k_\alpha := k_\beta]k_\beta \\ &\equiv ([\alpha]t)^\circ[k_\alpha := k_\beta]\end{aligned}$$

2. Property 2: here we have $(t[\alpha := \beta\ (\Box s)])^\circ \twoheadrightarrow t^\circ[k_\alpha := \lambda l.ls^\circ k_\beta]$ by the induction hypothesis, so:

$$\begin{aligned}(([\alpha]t)[\alpha := \beta\ (\Box s)])^\circ &\equiv ([\beta]t[\alpha := \beta\ (\Box s)]s)^\circ \\ &\equiv (\lambda k.t[\alpha := \beta\ (\Box s)]^\circ(\lambda l.ls^\circ k))k_\beta \\ &\to t[\alpha := \beta\ (\Box s)]^\circ(\lambda l.ls^\circ k_\beta) \\ &\twoheadrightarrow t[k_\alpha := \lambda l.ls^\circ k_\beta]^\circ(\lambda l.ls^\circ k_\beta) \\ &\equiv ([\alpha]t)^\circ[k_\alpha := \lambda l.ls^\circ k_\beta]\end{aligned}$$

3. Property 3 is similar to property 2. □

Lemma 3.5.9. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda^{\boldsymbol{T}}$ preserves equality. That is, given $\lambda_\mu^{\mathbf{T}}$-terms t_1 and t_2 such that $t_1 = t_2$, then ${t_1}^\circ = {t_2}^\circ$.*

Proof. By induction on $t_1 \to t_2$.

1. Let $(\lambda x.t)r \to t[x := r]$. Now:

$$\begin{aligned}((\lambda x.t)r)^\circ &\equiv \lambda k.(\lambda n.n(\lambda x.t^\circ))\lambda l.lr^\circ k \\ &\twoheadrightarrow \lambda k.t^\circ[x := r^\circ]k && \text{(a)} \\ &= \lambda k.t[x := r]^\circ k && \text{(b)} \\ &= t[x := r]^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 3.5.7 and step (b) by Lemma 3.5.6.

2. The case $(\lambda\gamma.t)\rho \to t[\gamma := \rho]$ is similar to the preceding case.

3. Let $(\mu\alpha.c)s \to \mu\alpha c[\alpha := \alpha\ (\Box s)]$. Now:

$$\begin{aligned}((\mu\alpha.c)s)^\circ &\equiv \lambda k.(\lambda k_\alpha.c^\circ)\lambda l.ls^\circ k \\ &\twoheadrightarrow \lambda k.c^\circ[k_\alpha := \lambda l.ls^\circ k] \\ &= \lambda k_\alpha.c[\alpha := \alpha\ (\Box s)]^\circ && \text{(a)} \\ &\equiv (\mu\alpha.c[\alpha := \alpha\ (\Box s)])^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 3.5.8.

4. The case $(\mu\alpha.c)\rho \to \mu\alpha c[\alpha := \alpha\ (\Box\rho)]$ is similar to the preceding case.

5. Let $\mu\alpha.[\alpha]t \to t$. Now:

$$\begin{aligned}(\mu\alpha.[\alpha]t)^\circ &\equiv \lambda k_\alpha.t^\circ k_\alpha && \text{(a)}\\ &= t^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 3.5.6.

6. Let $[\alpha]\mu\beta.c \to c[\beta := \alpha\ \Box]$. Now:

$$\begin{aligned}([\alpha]\mu\beta.c)^\circ &\equiv (\lambda k_\beta.c^\circ)k_\alpha\\ &\to c^\circ[k_\beta := k_\alpha] && \text{(a)}\\ &= c[\beta := \alpha\ \Box]^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 3.5.8. □

Unfortunately, our CPS-translation does not preserve reduction. For example, we have $(\mu\alpha.[\alpha]x)y \to \mu\alpha.[\alpha]xy$, but not:

$$\lambda k.(\lambda k_\alpha.(\lambda h.xh)\,k_\alpha)\,(\lambda l.l(\lambda h.yh)k) \twoheadrightarrow \lambda k_\alpha.(\lambda k.(\lambda h.xh)\,(\lambda l.l(\lambda h.yh)k))\,k_\alpha$$

This is caused by the so called *administrative reductions*, which also appear in Plotkin's CPS-translation. To repair this issue Plotkin introduced the *colon translation* $t : K$ [Plo75]. Here, $t^\star = \lambda k.t : k$ is the result of contracting all administrative redexes in t°. We can adapt Plotkin's colon translation for λ_μ as follows.

Definition 3.5.10. *Given a $\lambda_\mu^{\mathbf{2}}$-term t, then the* CPS-translation $t^\star$ *is mutually inductively defined with the* colon translation $t : K$ *as follows.*

$$\begin{aligned}t^\star &:= t : (\lambda k.t : k)\\ ([\alpha]t)^\star &:= t : k_\alpha\\ x : K &:= xK\\ (\lambda x.t) : K &:= K(\lambda x.t^\star)\\ (tr) : K &:= t : (\lambda l.lr^\star K)\\ (\lambda\gamma.t) : K &:= K(\lambda\gamma.t^\star)\\ (t\rho) : K &:= t : (\lambda l.l\rho^\bullet K)\\ (\mu\alpha.c) : K &:= c^\star[k_\alpha := K]\end{aligned}$$

Here, k_α is a fresh λ-variable for each μ-variable α.

However, a straightforward adaption of the colon translation, like ours, does not work for the λ_μ-calculus [IN06]. Our translation is merely weakly reduction preserving, that is, one reduction step may be translated in zero reduction steps. For example, we have $(\mu\alpha.[\alpha]x)y \to \mu\alpha.[\alpha]xy$, but:

$$\begin{aligned}(\mu\alpha.[\alpha]x)y : K &\equiv \mu\alpha.[\alpha]x : \lambda l.l(\lambda h.yh)K\\ &\equiv (x : k_\alpha)[k_\alpha := \lambda l.l(\lambda h.yh)K]\\ &\equiv x\,(\lambda l.l(\lambda h.yh)K)\\ &\equiv (x : \lambda l.l(\lambda h.yh)k_\alpha)[k_\alpha := K]\\ &\equiv (xy : k_\alpha)[k_\alpha := K]\\ &\equiv (\mu\alpha.[\alpha]xy) : K\end{aligned}$$

In order to repair this issue, Ikeda and Kakazawa described an alternative CPS-translation [IN06], in which terms are not only parametrized by continuations but also by so called *garbage terms*. However, we will not go into the details here, and refer to [IN06] for the niceties of their translation.

On a completely different track, we refer to [dG94] and [Fuj03] for complete CPS-translations of λ_μ. These translations are too involved to be discussed in this thesis.

Chapter 4

The λ_μ-calculus with arithmetic

In the previous chapters we have discussed various first-order typed λ-calculi with control operators, but so far, none of these systems contained basic types like the natural numbers or lists as primitives. Although the natural numbers can be encoded by first-order Church numerals in $\lambda\!\!\rightarrow$, we have already remarked that this does not yield much expressive power. The simply typed λ_μ-calculus does not extend this class of functions since each simply typed λ_μ-term can be translated back into $\lambda\!\!\rightarrow$ by CPS.

A well known extension of $\lambda\!\!\rightarrow$ is Gödel's $\mathbf{T}$, this system contains the natural numbers as a primitive type and primitive recursion as a primitive construct. Instead of just the extended polynomials, all functions that are provably recursive in first-order arithmetic are definable in it [SU06]. So, since arithmetic makes $\lambda\!\!\rightarrow$ much stronger, we might wonder whether we could add arithmetic to λ-calculi with control as well. But to the author's surprise and knowledge there is little evidence of research in which a typed λ-calculus is extended with both arithmetic and control operators. In the following paragraphs we will summarize relevant research.

Murthy considered a system with control operators, arithmetic, products and sums in his PhD thesis [Mur90]. But his system uses the control operators $\mathcal{C}$ and $\mathcal{A}$ and the semantics of these operators is specified by evaluation contexts rather than local reduction rules. He furthermore mainly considered CPS-translations and did not prove properties like confluence or strong normalization of his extended system. Crolard and Polonowski have considered a version of Gödel's $\mathbf{T}$ with products and `call/cc` in [CP09]. Unfortunately the semantics is presented by CPS-translations instead of a direct specification. Therefore properties like confluence and strong normalization are trivial because they hold for the target system already.

Barthe and Uustalu have worked on CPS-translations for inductive and coinductive types [BU02]. Their work includes a system with a primitive for iteration over the natural numbers and the control operator Δ. Unfortunately only some properties of CPS-translations are proven.

Furthermore, in [RS94], Rehof and Sørensen have described an extension

of the λ_Δ-calculus with basic constants and functions. In their extension a function $\delta : \texttt{function\_constant} \times \texttt{basic\_constant} \to \texttt{value}$ is used for the reduction rules $f\ b \to \delta(f, b)$ and $f\ \Delta x.t \to \Delta x.t[x := \lambda y.x\ (f\ y)]$. They have proven that their system is confluent, but unfortunately it is very limited. For example the primitive recursor `nrec` takes terms, rather than basic constants, as its arguments, hence this extension cannot be used to define primitive recursion.

In this chapter we will present a Gödel's **T** version of the $\lambda\mu$-calculus and prove basic properties like subject reduction, confluence and strong normalization. In the Section 4.1 we will describe Gödel's **T** and some of its important properties. Readers that are already well known with Gödel's **T** can safely skip the next section and continue reading in Section 4.2

4.1 Gödel's T

Gödel's **T** (henceforth $\lambda^{\mathbf{T}}$) was invented by Kurt Gödel to prove the consistency of Peano Arithmetic [SU06]. It arises from $\lambda\!\to$ by addition of a base type for natural numbers and a construct for primitive recursion.

Definition 4.1.1. *The* types *of $\lambda^{\mathbf{T}}$ are built from a basic type (the natural numbers) and an implication arrow ($\to$) as follows.*

$$\rho, \delta ::= \mathtt{N} \mid \rho \to \delta$$

Definition 4.1.2. *The* terms *of the $\lambda^{\mathbf{T}}$ are inductively defined over an infinite set of* λ-variables $(x, y, \ldots)$ *as follows.*

$$\begin{aligned} t, r, s ::= {} & x \mid \lambda x : \rho.r \mid ts \\ & \mid 0 \mid \mathtt{S}t \mid \mathtt{nrec}_\rho\ r\ s\ t \end{aligned}$$

Here, ρ ranges over $\lambda^{\mathbf{T}}$-types.

As one would image, the terms 0, `S` and `nrec` denote zero, the successor function and primitive recursion over the natural numbers, respectively. Many presentations of Gödel's **T** [GTL89, for example] also include a basic type for the booleans. However a boolean type is superfluous as the booleans `ff` and `tt` can be represented by the natural numbers 0 and `S0`, respectively, and the conditional `bcase` $r\ s\ t$ can be represented by `nrec` $r\ (\lambda x h.s)\ t$. Hence we will omit a boolean type so as to keep our system as simple as possible.

As usual, we let $\mathrm{FV}(t)$ denote the set of free variables of a term t and we generalize the operation of capture avoiding substitution $t[x := r]$ of r for x in t to $\lambda^{\mathbf{T}}$-terms in the obvious way.

Definition 4.1.3. *A* $\lambda^{\mathbf{T}}$-typing judgment $\Gamma \vdash t : \rho$ *denotes that a term t has type ρ in an environment Γ. The derivation rules for such judgments are shown in Figure 4.1.*

Definition 4.1.4. *Reduction $t \to t'$ on $\lambda^{\mathbf{T}}$-terms t and t' is defined as the compatible closure of the rules displayed in Figure 4.2. As usual, $\twoheadrightarrow$ denotes the reflexive/transitive closure and $=$ denotes the reflexive/symmetric/transitive closure.*

$$\frac{x : \rho \in \Gamma}{\Gamma \vdash x : \rho}\ \text{(a) var} \qquad \frac{\Gamma, x : \rho \vdash t : \delta}{\Gamma \vdash \lambda x : \rho.t : \rho \to \delta}\ \text{(b) lambda} \qquad \frac{\Gamma \vdash t : \rho \to \delta \qquad \Gamma \vdash s : \rho}{\Gamma \vdash ts : \delta}\ \text{(c) app}$$

$$\Gamma \vdash 0 : \mathtt{N}\ \text{(d) zero} \qquad \frac{\Gamma \vdash t : \mathtt{N}}{\Gamma \vdash \mathtt{S}t : \mathtt{N}}\ \text{(e) suc} \qquad \frac{\Gamma \vdash r : \rho \qquad \Gamma \vdash s : \mathtt{N} \to \rho \to \rho \qquad \Gamma \vdash t : \mathtt{N}}{\Gamma \vdash \mathtt{nrec}_\rho\ r\ s\ t : \rho}\ \text{(f) nrec}$$

Figure 4.1: The typing rules of $\lambda^{\mathbf{T}}$.

$$\begin{array}{rcl} (\lambda x.t)r & \to_\beta & t[x := r] \\ \mathtt{nrec}\ r\ s\ 0 & \to_0 & r \\ \mathtt{nrec}\ r\ s\ (\mathtt{S}t) & \to_{\mathtt{S}} & s\ t\ (\mathtt{nrec}\ r\ s\ t) \end{array}$$

Figure 4.2: The reduction rules of $\lambda^{\mathbf{T}}$.

Although we do not specify a specific reduction strategy it is obviously possible to create a call-by-name and call-by-value version of $\lambda^{\mathbf{T}}$. Yet it is interesting to remark that in a call-by-value version of $\lambda^{\mathbf{T}}$ calculating the predecessor takes at least linear time while in a call-by-name version the predecessor can be calculated in constant time [CF98].

Fortunately, despite the additional features of $\lambda^{\mathbf{T}}$, the important properties of $\lambda\!\to$ are preserved.

Lemma 4.1.5. *$\lambda^{\mathbf{T}}$ satisfies subject reduction.*

Proof. First we have to prove a similar substitution lemma as we have proven for $\lambda\!\to$ (Lemma 2.3.7). Then we have to prove that all reduction rules preserve typing. We treat some interesting reduction rules.

1. The $\to_0$-rule:

$$\frac{r : \rho \qquad s : \mathtt{N} \to \rho \to \rho \qquad t : \mathtt{N}}{\mathtt{nrec}_\rho\ r\ s\ t : \rho} \quad \to_0 \quad r : \rho$$

2. The $\to_{\mathtt{S}}$-rule:

$$\frac{r : \rho \qquad s : \mathtt{N} \to \rho \to \rho \qquad \dfrac{t : \mathtt{N}}{\mathtt{S}t : \mathtt{N}}}{\mathtt{nrec}_\rho\ r\ s\ \mathtt{S}t : \rho} \quad \to_{\mathtt{S}} \quad \frac{\dfrac{s : \mathtt{N} \to \rho \to \rho \quad t : \mathtt{N}}{s\ t : \rho \to \rho} \qquad \dfrac{r : \rho \quad s : \mathtt{N} \to \rho \to \rho \quad t : \mathtt{N}}{\mathtt{nrec}_\rho\ r\ s\ t : \rho}}{s\ t\ (\mathtt{nrec}_\rho\ r\ s\ t) : \rho}$$

□

Lemma 4.1.6. *$\lambda^{\mathbf{T}}$ is confluent.*

Proof. This is proven in [GTL89]. □

Lemma 4.1.7. *$\lambda^{\mathbf{T}}$ is strongly normalizing.*

Proof. This is proven in [GTL89]. □

Because it is convenient to be able to talk about a term representing an actual natural number we introduce the following notation.

Notation 4.1.8. $\underline{n} := \mathtt{S}^n 0$

Now we introduce the notion of values for $\lambda^{\mathbf{T}}$ and prove that each closed term that is in normal form is a value.

Definition 4.1.9. Values *are inductively defined as follows.*

$$v, w ::= 0 \mid \mathtt{S}v \mid \lambda x.r$$

Lemma 4.1.10. *Given a term t that is in normal form and such that $\vdash t : \rho$, then:*

1. *If $\rho = \mathtt{N}$, then $t \equiv \underline{n}$ for some $n \in \mathbb{N}$.*
2. *If $\rho = \gamma \to \delta$, then $t \equiv \lambda x.r$ for a variable x and term r.*

Proof. By induction on the derivation $\vdash t : \rho$.

(var) Let $\vdash x : \rho$ with $x : \rho \in \emptyset$. Now we obtain a contradiction since $x : \rho \notin \emptyset$.

(λ) Let $\vdash \lambda x.r : \gamma \to \delta$. Now we are immediately done.

(app) Let $\vdash rs : \rho$ with $\vdash r : \delta \to \rho$ and $\vdash s : \delta$. Now we have $r \equiv \lambda x.r'$ by the induction hypothesis. But therefore we obtain a contradiction since rs should be in normal form.

(zero) Let $\vdash 0 : \mathtt{N}$. Now we are immediately done because $0 \equiv \underline{0}$.

(suc) Let $\vdash \mathtt{S}t : \mathtt{N}$ with $\vdash t : \mathtt{N}$. Now we have $t \equiv \underline{n}$ for some $n \in \mathbb{N}$ by the induction hypothesis, so $\mathtt{S}t \equiv \mathtt{S}\underline{n} \equiv \underline{n+1}$.

(nrec) Let $\vdash \mathtt{nrec}\ r\ s\ t : \rho$ with $\vdash t : \mathtt{N}$. Now we have $t \equiv \underline{n}$ for some $n \in \mathbb{N}$ by the induction hypothesis. But therefore we obtain a contradiction since $\mathtt{nrec}\ r\ s\ t$ should be in normal form. □

Moreover, as the following theorem indicates, it turns out that $\lambda^{\mathbf{T}}$ has quite some expressive power.

Definition 4.1.11. *A function $f : \mathbb{N}^n \to \mathbb{N}$ is* representable *in $\lambda^{\mathbf{T}}$ if there is a term t such that:*

$$t\ \underline{m_1} \dots \underline{m_n} = \underline{f(m_1, \dots, m_n)}$$

Theorem 4.1.12. *The functions definable in $\lambda^{\mathbf{T}}$ are exactly the functions that are provably recursive in first-order arithmetic*[1]*.*

Proof. This is proven in [SU06]. □

[1] Here we are allowed to say either Peano Arithmetic (PA) or Heyting Arithmetic (HA), because a function is provably recursive in PA iff it is probably recursive in HA [SU06].

4.2 The $\lambda_\mu^{\mathbf{T}}$-calculus

In this section we will present a Gödel's **T** variant of Parigot's λ_μ-calculus (henceforth $\lambda_\mu^{\mathbf{T}}$).

Definition 4.2.1. *The* terms *and* commands *of* $\lambda_\mu^{\mathbf{T}}$ *are mutually inductively defined over an infinite set of* λ-variables $(x, y, \ldots)$ *and* μ-variables $(\alpha, \beta, \ldots)$ *as follows.*

$$\begin{aligned} t, r, s ::= {} & x \mid \lambda x : \rho.r \mid ts \mid \mu\alpha : \rho.c \\ & \mid 0 \mid \mathtt{S}t \mid \mathtt{nrec}_\rho\ r\ s\ t \\ c, d ::= {} & [\alpha]t \end{aligned}$$

Here, ρ *ranges over* $\lambda^{\mathbf{T}}$*-types (Definition 4.1.1).*

As usual, we let $\mathrm{FV}(t)$ and $\mathrm{FCV}(t)$ denote the set of free λ-variables and μ-variables of a term t, respectively. Moreover, the operation of capture avoiding substitution $t[x := r]$ of r for x in t generalizes to $\lambda_\mu^{\mathbf{T}}$-terms in the obvious way.

Definition 4.2.2. *A* $\lambda_\mu^{\mathbf{T}}$-typing judgment $\Gamma; \Delta \vdash t : \rho$ *denotes that a term* t *has type* ρ *in an environment of* λ*-variables* Γ *and an environment of* μ*-variables* Δ*. A typing judgment* $\Gamma; \Delta \vdash c : \perp\!\!\!\perp$ *denotes that a command* c *is typable in an environment of* λ*-variables* Γ *and an environment of* μ*-variables* Δ*. The derivation rules for such judgments are mutually recursively defined and shown in Figure 4.3.*

$$\frac{x : \rho \in \Gamma}{\Gamma; \Delta \vdash x : \rho}\ \text{(a) axiom} \qquad \frac{\Gamma, x : \rho; \Delta \vdash t : \delta}{\Gamma; \Delta \vdash \lambda x : \rho.t : \rho \to \delta}\ \text{(b) lambda} \qquad \frac{\Gamma; \Delta \vdash t : \rho \to \delta \qquad \Gamma; \Delta \vdash s : \rho}{\Gamma; \Delta \vdash ts : \delta}\ \text{(c) app}$$

$$\Gamma \vdash 0 : \mathtt{N}\ \text{(d) zero} \qquad \frac{\Gamma \vdash t : \mathtt{N}}{\Gamma \vdash \mathtt{S}t : \mathtt{N}}\ \text{(e) suc} \qquad \frac{\Gamma \vdash r : \rho \qquad \Gamma \vdash s : \mathtt{N} \to \rho \to \rho \qquad \Gamma \vdash t : \mathtt{N}}{\Gamma \vdash \mathtt{nrec}_\rho\ r\ s\ t : \rho}\ \text{(f) nrec}$$

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash c : \perp\!\!\!\perp}{\Gamma; \Delta \vdash \mu\alpha : \rho.c : \rho}\ \text{(g) activate} \qquad \frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash [\alpha]t : \perp\!\!\!\perp}\ \text{(h) passivate}$$

Figure 4.3: The typing rules of $\lambda_\mu^{\mathbf{T}}$.

In order to extend the notion of structural substitution we have to extend the notion of contexts to the language of $\lambda_\mu^{\mathbf{T}}$-terms.

Definition 4.2.3. *A* $\lambda_\mu^{\mathbf{T}}$-context *is defined as follows.*

$$E ::= \Box \mid Et \mid \mathtt{S}E \mid \mathtt{nrec}\ r\ s\ E$$

Now we generalize the notion of substitution $E[s]$ of s for the hole $\Box$ in E and the notion of structural substitution $t[\alpha := \beta E]$ of βE for α in t in a straightforward way.

Definition 4.2.4. *Reduction $t \to t'$ on $\lambda_\mu^{\mathbf{T}}$-terms t and t' is defined as the compatible closure of the rules displayed in Figure 4.4. As usual, $\twoheadrightarrow^+$ denotes the transitive closure, $\twoheadrightarrow$ denotes the reflexive/transitive closure and $=$ denotes the reflexive/symmetric/transitive closure.*

$$\begin{array}{rll}
(\lambda x.t)r & \to_\beta & t[x := r] \\
\mathtt{S}(\mu\alpha.c) & \to_{\mu\mathtt{S}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)] \\
(\mu\alpha.c)s & \to_{\mu R} & \mu\alpha.c[\alpha := \alpha\ (\Box s)] \\
\mu\alpha.[\alpha]t & \to_{\mu\eta} & t \qquad \text{provided that } \alpha \notin \mathrm{FCV}(t) \\
[\alpha]\mu\beta.c & \to_{\mu i} & c[\beta := \alpha\ \Box] \\
\mathtt{nrec}\ r\ s\ 0 & \to_0 & r \\
\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n}) & \to_{\mathtt{S}} & s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n}) \\
\mathtt{nrec}\ r\ s\ (\mu\alpha.c) & \to_{\mu\mathtt{N}} & \mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)]
\end{array}$$

Figure 4.4: The reduction rules of $\lambda_\mu^{\mathbf{T}}$.

Lemma 4.2.5. *The $\lambda_\mu^{\mathbf{T}}$-calculus satisfies subject reduction.*

Proof. First we have to prove a similar substitution lemma as we have proven for $\lambda\!\to$ (Lemma 2.3.7) and structural substitution lemma as we have proven for λ_μ (Lemma 3.3.15). Then we have to prove that all reduction rules preserve typing. We treat some interesting reduction rules.

1. The $\to_{\mu\mathtt{S}}$-rule:

$$\dfrac{\dfrac{\dfrac{c : \perp\!\!\!\perp}{\mu\alpha.c : \mathtt{N}}}{\mathtt{S}(\mu\alpha.c) : \mathtt{N}}}{} \quad \to_{\mu\mathtt{S}} \quad \dfrac{c[\alpha := \alpha\ (\mathtt{S}\Box)] : \perp\!\!\!\perp}{\mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)] : \mathtt{N}}$$

   Here we have $c[\alpha := \alpha\ (\mathtt{S}\Box)] : \perp\!\!\!\perp$ by the structural substitution lemma.

2. The $\to_{\mu\mathtt{N}}$-rule:

$$\dfrac{r : \rho \quad s : \mathtt{N} \to \rho \to \rho \quad \dfrac{c : \perp\!\!\!\perp}{\mu\alpha.c : \mathtt{N}}}{\mathtt{nrec}\ r\ s\ (\mu\alpha.c) : \rho} \quad \to_{\mu\mathtt{N}} \quad \dfrac{c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)] : \perp\!\!\!\perp}{\mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)] : \rho}$$

   It remains to prove that $\Gamma'; \Delta' \vdash c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)] : \perp\!\!\!\perp$. We proceed by applying the structural substitution lemma, so given contexts $\Gamma' \supseteq \Gamma$, $\Delta' \supseteq \Delta$ and a derivation $\Gamma'; \Delta' \vdash t : \mathtt{N}$, then we have to prove that $\Gamma'; \Delta' \vdash \mathtt{nrec}\ r\ s\ t : \delta$. The required result is shown below.

$$\dfrac{\dfrac{\Gamma; \Delta \vdash r : \rho}{\Gamma'; \Delta' \vdash r : \rho} \quad \dfrac{\Gamma; \Delta \vdash s : \mathtt{N} \to \rho \to \rho}{\Gamma'; \Delta' \vdash s : \mathtt{N} \to \rho \to \rho} \quad \Gamma'; \Delta' \vdash t : \mathtt{N}}{\Gamma'; \Delta' \vdash \mathtt{nrec}\ r\ s\ t : \rho}$$

□

Notice that the $\to_{\mathtt{S}}$-rule, contrary to the corresponding rule of $\lambda^{\mathbf{T}}$ (Definition 4.1.4), restricts the reduction order in way similar to call-by-value. In order to apply this rule, the third argument of $\mathtt{nrec}$ should be reduced to an actual numeral. This restriction ensures that primitive recursion is not performed on terms that might reduce to a term of the shape $\mu\alpha.c$. If we omit this restriction we lose confluence. We illustrate this by considering a variant of our system with the following rule instead.

$$\mathtt{nrec}\ r\ s\ (\mathtt{S}t) \to_{\mathtt{S}'} s\ t\ (\mathtt{nrec}\ r\ s\ t)$$

Now we can reduce the term $t \equiv \mu\alpha.[\alpha]\mathtt{nrec}\ 0\ (\lambda xh.\underline{2})\ (\mathtt{S}\Theta[\alpha]\underline{4})$ to two distinct normal forms:

$$\begin{aligned} t &\equiv \mu\alpha.[\alpha]\mathtt{nrec}\ 0\ (\lambda xh.\underline{2})\ (\mathtt{S}\Theta[\alpha]\underline{4}) \\ &\to \mu\alpha.[\alpha]\mathtt{nrec}\ 0\ (\lambda xh.\underline{2})\ \Theta[\alpha]\underline{4} \\ &\to \mu\alpha.[\alpha]\Theta[\alpha]\underline{4} \\ &\to \mu\alpha.[\alpha]\underline{4} \to \underline{4} \end{aligned}$$

And:

$$\begin{aligned} t &\equiv \mu\alpha.[\alpha]\mathtt{nrec}\ 0\ (\lambda xh.\underline{2})\ (\mathtt{S}\Theta[\alpha]\underline{4}) \\ &\to \mu\alpha.[\alpha](\lambda xh.\underline{2})\ (\Theta[\alpha]\underline{4})\ (\mathtt{nrec}\ 0\ (\lambda xh.\underline{2})\ \Theta[\alpha]\underline{4}) \\ &\twoheadrightarrow \mu\alpha.[\alpha]\underline{2} \to \underline{2} \end{aligned}$$

Alternatively, in order to obtain a confluent system, it is possible to remove the $\to_{\mathtt{S}}$-rule while retaining the unrestricted $\to_{\mu\mathtt{S}'}$-rule. However, then we can construct closed terms $t : \mathtt{N}$ that are in normal form but such that $t \not\equiv \underline{n}$. An example of such a term is $\mu\alpha.[\alpha]\mathtt{S}\mu\beta.[\alpha]0$.

Lemma 4.2.6. *Given a term t that is in normal and such that $;\Delta \vdash t : \rho$, then:*

1. *If $\rho = \mathtt{N}$, then $t \equiv \underline{n}$ or $t \equiv \mu\alpha.[\beta]\underline{n}$ for some $n \in \mathbb{N}$.*
2. *If $\rho = \gamma \to \delta$, then $t \equiv \lambda x.r$ or $t \equiv \mu\alpha.[\beta]\lambda x.r$ for some variable x and term r.*

Proof. By induction on the derivation $;\Delta \vdash t : \rho$.

(var) Let $;\Delta \vdash x : \rho$ with $x : \rho \in \emptyset$. Now we obtain a contradiction since $x : \rho \notin \emptyset$.

(λ) Let $;\Delta \vdash \lambda x.r : \gamma \to \delta$. Now we are immediately done.

(app) Let $;\Delta \vdash rs : \rho$ with $;\Delta \vdash r : \delta \to \rho$ and $;\Delta \vdash s : \delta$. Now by the induction hypothesis we have $r \equiv \lambda x.r'$ or $r \equiv \mu\alpha.[\beta]\lambda x.r'$. But since rs should be in normal form we obtain a contradiction.

(zero) Let $;\Delta \vdash 0 : \mathtt{N}$. Now we are immediately done because $0 \equiv \underline{0}$.

(suc) Let $;\Delta \vdash \mathtt{S}t : \mathtt{N}$ with $;\Delta \vdash t : \mathtt{N}$. Now we have $t \equiv \underline{n}$ or $t \equiv \mu\alpha.[\beta]\underline{n}$ for some $n \in \mathbb{N}$ by the induction hypothesis. In the former case we are immediately done, in the latter case we obtain a contradiction because the $\to_{\mu S}$-rule can be applied.

(nrec) Let $;\Delta \vdash \mathtt{nrec}\ r\ s\ t : \rho$ with $;\Delta \vdash t : \mathtt{N}$. Now we have $t \equiv \underline{n}$ or $t \equiv \mu\alpha.[\beta]\underline{n}$ for some $n \in \mathbb{N}$ by the induction hypothesis. But in both cases we obtain a contradiction because the reduction rules $\rightarrow_{\mu 0}$, $\rightarrow_{\mu S}$ and $\rightarrow_{\mu \mathtt{N}}$ can be applied, respectively.

(act/pas) Let $;\Delta \vdash \mu\alpha.[\beta]t : \rho$ with $;\Delta, \alpha : \rho \vdash t : \sigma$ and $\beta : \sigma \in (\Delta, \alpha : \rho)$. We distinguish the following cases.

(a) Suppose that $\sigma = \mathtt{N}$. Now we have $t \equiv \underline{n}$ or $t \equiv \mu\alpha.[\beta]\underline{n}$ for some $n \in \mathbb{N}$ by the induction hypothesis. In the former case we are immediately done. In the latter case we obtain a contradiction because the $\rightarrow_{\mu i}$-rule can be applied.

(b) Suppose that $\sigma = \gamma \rightarrow \delta$. Now we have $t \equiv \lambda x.r'$ or $t \equiv \mu\alpha.[\beta]\lambda x.r'$ by the induction hypothesis. Again in the former case we are done. In the latter case we obtain a contradiction because the $\rightarrow_{\mu i}$-rule can be applied. □

Lemma 4.2.7. *Given a term t that is in normal form and such that $;\vdash t : \mathtt{N}$, then $t \equiv \underline{n}$ for some $n \in \mathbb{N}$.*

Proof. By Lemma 4.2.6 we obtain that $t \equiv \underline{n}$ or $t \equiv \mu\alpha.[\beta]\underline{n}$ for some $n \in \mathbb{N}$. In the former case we are immediately done and in the latter case we know that $\beta = \alpha$ since t is closed for μ-variables, hence we can apply the $\rightarrow_{\mu\eta}$-rule and obtain a contradiction. □

In the remainder of this section we define some additional notions and prove some meta theoretical properties. These notions are essential for our proof of confluence (Section 4.3) and strong normalization (Section 4.4).

Definition 4.2.8. *A* singular $\lambda_\mu^{\mathbf{T}}$-context *is a context of the following shape.*

$$E^s ::= \Box t \mid \mathtt{S}\Box \mid \mathtt{nrec}\ r\ s\ \Box$$

Using the notion of a singular context it is possible to replace the reduction rules $\rightarrow_{\mu\mathtt{S}}$, $\rightarrow_{\mu R}$ and $\rightarrow_{\mu\mathtt{N}}$ by one single rule (adapted from [FH92]).

$$E^s[\mu\alpha.c] \rightarrow \mu\alpha.c[\alpha := \alpha E^s]$$

As a direct consequence we have the following lemma.

Lemma 4.2.9. *Given a command c and context E, then we have:*

$$E[\mu\alpha.c] \twoheadrightarrow \mu\alpha.c[\alpha := \alpha E]$$

Proof. By induction on the structure of E. □

Definition 4.2.10. *Given contexts E and F, then a context EF is defined as:*

$$\begin{aligned} \Box F &:= F \\ (Et)F &:= (EF)t \\ (\mathtt{S}E)F &:= \mathtt{S}(EF) \\ (\mathtt{nrec}\ r\ s\ E)F &:= \mathtt{nrec}\ r\ s\ (EF) \end{aligned}$$

Lemma 4.2.11. *Given contexts E and F and a term t, then we have:*

$$E[F[t]] \equiv EF[t]$$

Proof. By induction on the structure of E. □

The notion of of free λ-variables, free μ-variables, substitution and structural substitution are generalized to contexts in the obvious way. The following lemma states some essential properties of substitution.

Lemma 4.2.12. *(Structural) substitution satisfies the following properties.*

1. $\mu\alpha.c \equiv \mu\beta.c[\alpha := \beta\ \Box]$ *provided that* $\beta \notin \mathrm{FCV}(c)$.
2. $t[x := r][y := s] \equiv t[y := s][x := r[y := s]]$ *provided that* $x \neq y$, $x \notin \mathrm{FV}(s)$.
3. $t[x := r][\beta := \beta F] \equiv t[\beta := \beta F][x := r[\beta := \beta F]]$ *provided that* $x \notin \mathrm{FV}(F)$.
4. $t[\alpha := \alpha' E][y := s] \equiv t[y := s][\alpha := \alpha' E[y := s]]$ *provided that* $\alpha \notin \mathrm{FCV}(s)$.
5. $t[\alpha := \alpha' E][\beta := \beta' F] \equiv t[\beta := \beta' F][\alpha := \alpha' E[\beta := \beta' F]]$ *provided that* $\alpha \neq \beta$, $\alpha' \neq \beta$, $\alpha \neq \beta'$ *and* $\alpha \notin \mathrm{FCV}(F)$.
6. $t[\alpha := \beta E][\beta := \beta' F] \equiv t[\beta := \beta' F][\alpha := \beta' F(E[\beta := \beta' F])]$ *provided that* $\alpha \neq \beta$, $\alpha \neq \beta'$ *and* $\alpha \notin \mathrm{FCV}(F)$.
7. $t[\alpha := \beta EF] \equiv t[\alpha := \alpha F][\alpha := \beta E]$ *provided that* $\alpha \notin \mathrm{FCV}(F)$.
8. $t[\alpha := \beta EF] \equiv t[\alpha := \beta F][\beta := \beta E]$ *provided that* $\beta \notin \mathrm{FCV}(F) \cup \mathrm{FCV}(t)$.

Proof. All properties are proven by induction on the structure of t. □

4.3 Confluence of $\lambda_\mu^{\mathbf{T}}$

To prove confluence one typically uses the notion of *parallel reduction* by Tait and Martin-Löf. A parallel reduction relation $\Rightarrow$ intuitively allows to contract multiple redexes in a term simultaneously. Because $\Rightarrow$ can be defined inductively and is preserved under substitution it is straightforward (by considering all critical pairs) to prove that it is confluent. Also, since $\Rightarrow$ is defined in such a way that $t \Rightarrow^* t'$ iff $t \twoheadrightarrow t'$, we obtain confluence of $\twoheadrightarrow$ as well. To streamline proving confluence of $\Rightarrow$ one could define the *complete development* $t^\diamond$ of each term t, which is obtained by contracting all redexes in t. Now it suffices to prove that $t \Rightarrow t'$ implies $t' \Rightarrow t^\diamond$. Unfortunately, as observed in [Fuj97, BHF01], adopting the notion of parallel reduction in a straightforward way does not work for λ_μ. The resulting parallel reduction relation will only be weakly confluent and not confluent.

In this section we will focus on resolving this problem for $\lambda_\mu^{\mathbf{T}}$. For an extensive discussion of parallel reduction and its application to various systems we refer to [Tak95]. Firstly, we will present the straightforward parallel reduction relation, which is obtained by extending Parigot's parallel reduction reduction [Par92] to $\lambda_\mu^{\mathbf{T}}$.

Temporary Definition 4.3.1. Parallel reduction $t \Rightarrow t'$ on terms t and t' *is mutually inductively defined with* parallel reduction $c \Rightarrow c'$ on commands c and c' *as follows.*

(t1) $x \Rightarrow x$

(t2) $0 \Rightarrow 0$

(t3) *If* $t \Rightarrow t'$, *then* $\lambda x.t \Rightarrow \lambda x.t'$.

(t4.1) *If* $t \Rightarrow t'$ *and* $r \Rightarrow r'$, *then* $tr \Rightarrow t'r'$.

(t4.2) *If* $t \Rightarrow t'$, *then* $\mathtt{S}t \Rightarrow \mathtt{S}t'$.

(t4.3) *If* $r \Rightarrow r'$, $s \Rightarrow s'$ *and* $t \Rightarrow t'$, *then* $\mathtt{nrec}\ s\ r\ t \Rightarrow \mathtt{nrec}\ s'\ r'\ t'$.

(t5) *If* $t \Rightarrow t'$ *and* $r \Rightarrow r'$, *then* $(\lambda x.t)r \Rightarrow t'[x := r']$.

(t6.1) *If* $c \Rightarrow c'$, *then* $\mu\alpha.c \Rightarrow \mu\alpha.c'$.

(t6.2) *If* $c \Rightarrow c'$ *and* $s \Rightarrow s'$, *then* $(\mu\alpha.c)s \Rightarrow \mu\alpha.c'[\alpha := \alpha\ (\Box s')]$.

(t6.3) *If* $c \Rightarrow c'$, *then* $\mathsf{S}(\mu\alpha.c) \Rightarrow \mu\alpha.c'[\alpha := \alpha\ (\mathsf{S}\Box)]$.

(t6.4) *If* $r \Rightarrow r'$, $s \Rightarrow s'$ *and* $c \Rightarrow c'$, *then* $\mathsf{nrec}\ r\ s\ \mu\alpha.c \Rightarrow \mu\alpha.c'[\alpha := \alpha\ (\mathsf{nrec}\ r'\ s'\ \Box)]$.

(t7) *If* $t \Rightarrow t'$ *and* $\alpha \notin \mathrm{FCV}(t)$, *then* $\mu\alpha.[\alpha]t \Rightarrow t'$.

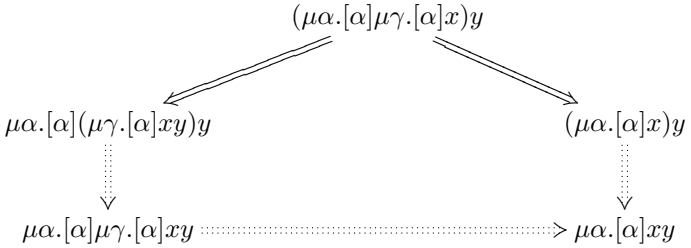
(t8) *If* $r \Rightarrow r'$, *then* $\mathtt{nrec}\ r\ s\ 0 \Rightarrow r'$.

(t9) *If* $r \Rightarrow r'$ *and* $s \Rightarrow s'$, *then* $\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n}) \Rightarrow s'\ \underline{n}\ (\mathtt{nrec}\ r'\ s'\ \underline{n})$.

(c1) *If* $t \Rightarrow t'$, *then* $[\alpha]t \Rightarrow [\alpha]t'$.

(c2) *If* $c \Rightarrow c'$, *then* $[\alpha]\mu\beta.c \Rightarrow c'[\beta := \alpha\Box]$.

Just as Parigot's original parallel reduction relation (as observed in [Fuj99]), our relation $\Rightarrow$ as in the preceding definition is not confluent. Let us (as in [BHF01]) consider the term $(\mu\alpha.[\alpha]\mu\gamma.[\alpha]x)y$, this term contains both a (t6.2) and a (c2)-redex. However, after contracting the (t6.2)-redex, we obtain the term $\mu\alpha.[\alpha](\mu\gamma.[\alpha]xy)y$, in which the (c2)-redex is stalled.

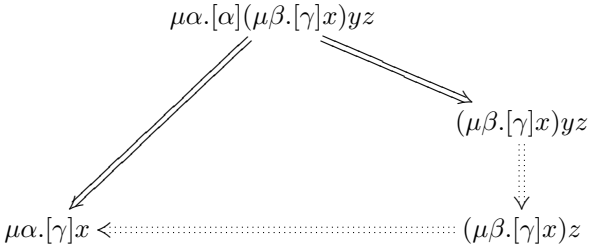


Although it is possible to prove that this relation is weakly confluent, weak confluence is not quite satisfactory. Of course, since $\lambda_\mu^{\mathbf{T}}$ is strongly normalizing (Theorem 4.4.34), it would give confluence by Newman's lemma. However, an untyped version of $\lambda_\mu^{\mathbf{T}}$ is obviously not strongly normalizing (consider the term Ω), hence we do not obtain confluence for it this way.

Baba, Hirokawa and Fujita noticed that this problem could be repaired by letting the (c2)-rule perform structural substitutions (t6.1-4) and renaming (c2) in one step [BHF01]. Their (c2)-rule is as follows.

(c2) If $c \Rightarrow c'$ and $E \Rightarrow E'$, then $[\alpha]E[\mu\beta.c] \Rightarrow c'[\beta := \alpha E']$.

Here E and E' are contexts and parallel reduction on contexts is defined by reducing all its components in parallel. Although they have shown that their relation for λ_μ without the (t7) rule is confluent, it is not confluent if the (t7) rule is included. Let us (as in [BHF01]) consider the term $\mu\alpha.[\alpha](\mu\beta.[\gamma]x)yz$.



In the conclusion of their work they suggested that this problem can be repaired by considering a series of structural substitutions (t6.1-4) as one step. This approach has been carried out successfully by Nakazawa for a call-by-value variant of λ_μ [Nak03]. However, Nakazawa did not use the notion of complete developments. We will repeat the approach suggested by Baba *et al.* for $\lambda_\mu^{\mathbf{T}}$ and use the notion of complete developments.

Definition 4.3.2. Parallel reduction $t \Rightarrow t'$ on terms t and t' *is mutually inductively defined with* parallel reduction $c \Rightarrow c'$ on commands c and c' *and* parallel reduction $E \Rightarrow E'$ on contexts E and E' *as follows.*

(t1) $x \Rightarrow x$

(t2) $0 \Rightarrow 0$

(t3) *If* $t \Rightarrow t'$, *then* $\lambda x.t \Rightarrow \lambda x.t'$.

(t4) *If* $t \Rightarrow t'$ *and* $E^s \Rightarrow E'$, *then* $E^s[t] \Rightarrow E'[t']$.

(t5) *If* $t \Rightarrow t'$ *and* $r \Rightarrow r'$, *then* $(\lambda x.t)r \Rightarrow t'[x := r']$.

(t6) *If* $c \Rightarrow c'$ *and* $E \Rightarrow E'$, *then* $E[\mu\alpha.c] \Rightarrow \mu\alpha.c'[\alpha := \alpha E']$.

(t7) *If* $t \Rightarrow t'$ *and* $\alpha \notin \mathrm{FCV}(t)$, *then* $\mu\alpha.[\alpha]t \Rightarrow t'$.

(t8) *If* $r \Rightarrow r'$, *then* $\mathtt{nrec}\ r\ s\ 0 \Rightarrow r'$.

(t9) *If* $r \Rightarrow r'$ *and* $s \Rightarrow s'$, *then* $\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n}) \Rightarrow s'\ \underline{n}\ (\mathtt{nrec}\ r'\ s'\ \underline{n})$.

(c1) *If* $t \Rightarrow t'$, *then* $[\alpha]t \Rightarrow [\alpha]t'$.

(c2) *If* $c \Rightarrow c'$ *and* $E \Rightarrow E'$, *then* $[\alpha]E[\mu\beta.c] \Rightarrow c'[\beta := \alpha E']$.

(E1) $\Box \Rightarrow \Box$

(E2) *If* $E \Rightarrow E'$ *and* $t \Rightarrow t'$, *then* $Et \Rightarrow E't'$.

(E3) *If* $E \Rightarrow E'$, *then* $\mathtt{S}E \Rightarrow \mathtt{S}E'$.

(E4) *If* $E \Rightarrow E'$, $r \Rightarrow r'$ *and* $s \Rightarrow s'$, *then* $\mathtt{nrec}\ r\ s\ E \Rightarrow \mathtt{nrec}\ r'\ s'\ E'$.

Moreover, $\Rightarrow^*$ *denotes the transitive closure of* $\Rightarrow$.

For notational reasons we specify the most of the forthcoming lemmas just for terms. Yet they are always mutually inductively proven for commands and contexts, hence these lemmas also hold for commands and contexts.

Lemma 4.3.3. *Parallel reduction is reflexive, that is* $t \Rightarrow t$ *for all terms* t.

Proof. By induction on t and the rules (t1-4), (t7), (c1) and (E1-4) □

Fact 4.3.4. *Given contexts* E *and* E' *such that* $E \Rightarrow E'$ *and terms* t *and* t' *such that* $t \Rightarrow t'$, *then* $E[t] \Rightarrow E'[t']$.

Fact 4.3.5. *Given a singular context* E^s *and a context* E' *such that* $E^s \Rightarrow E'$, *then* E' *is singular too.*

Fact 4.3.6. *Given terms* t *and* t' *such that* $t \Rightarrow t'$, *then* $x \in \mathrm{FV}(t')$ *implies* $x \in \mathrm{FV}(t)$ *and* $\alpha \in \mathrm{FCV}(t')$ *implies* $\alpha \in \mathrm{FCV}(t)$.

Lemma 4.3.7. *Parallel reduction is preserved under substitution, that is given terms* t *and* t' *such that* $t \Rightarrow t'$ *and terms* s *and* s' *such that* $s \Rightarrow s'$, *then:*

$$t[x := s] \Rightarrow t'[x := s']$$

Proof. By induction on $t \Rightarrow t'$. We treat some interesting cases.

(t1) Let $x \Rightarrow x$. Now $x[x := s] \equiv s \Rightarrow s' \equiv x[x := s']$ by assumption.

(t5) Let $(\lambda y.t)s \Rightarrow t'[y := r']$ with $t \Rightarrow t'$ and $r \Rightarrow r'$. Now we have $t[x := s] \Rightarrow t'[x := s']$ and $r[x := s] \Rightarrow r'[x := s']$ by the induction hypothesis, hence $(\lambda y.t[x := s])r[x := s] \Rightarrow t'[x := s'][y := r'[x := s']]$. By the Barendregt convention we have $y \neq x$ and $y \notin \mathrm{FV}(s)$, so $y \notin \mathrm{FV}(s')$ by Fact 4.3.6. Furthermore $t'[x := s'][y := r'[x := s']] \equiv t'[y := r'][x := s']$ by Lemma 4.2.12, so $((\lambda y.t)r)[x := s] \Rightarrow t'[y := r'][x := s']$. □

Lemma 4.3.8. *Parallel reduction is preserved under structural substitution, that is given terms* t *and* t' *such that* $t \Rightarrow t'$ *and contexts* E *and* E' *such that* $E \Rightarrow E'$, *then:*

$$t[\alpha := \beta E] \Rightarrow t'[\alpha := \beta E']$$

Proof. By induction on $t \Rightarrow t'$. We treat some interesting cases.

(t6) Let $F[\mu\gamma.c] \Rightarrow \mu\gamma.c'[\gamma := \gamma E']$ with $c \Rightarrow c'$ and $F \Rightarrow E'$. Moreover let $d \equiv c[\alpha := \beta E]$, $d' \equiv c'[\alpha := \beta E']$, $G = F[\alpha := \beta E]$ and $G' = E'[x := \beta E']$. Now we have $d \Rightarrow d'$ and $G \Rightarrow G'$ by the induction hypothesis, hence $G[\mu\gamma.d] \Rightarrow \mu\gamma.d'[\gamma := \gamma G']$. By the Barendregt convention we have $\gamma \neq \alpha$, $\gamma \neq \beta$ and $\gamma \notin F$, so $\gamma \notin \mathrm{FV}(E')$ by Fact 4.3.6. Furthermore we have $c'[\gamma := \gamma E'][\alpha := \beta E'] \equiv c'[\alpha := \beta E'][\gamma := \gamma G']$ by Lemma 4.2.12, so $(F[\mu\gamma.c])[\alpha := \beta E] \Rightarrow \mu\gamma.c'[\gamma := \alpha E'][\alpha := \beta E']$.

(c2) Let $[\alpha]F[\mu\gamma.c] \Rightarrow c'[\gamma := \alpha E']$ with $c \Rightarrow c'$ and $F \Rightarrow E'$. Moreover let $d \equiv c[\alpha := \beta E]$, $d' \equiv c'[\alpha := \beta E']$, $G = F[\alpha := \beta E]$ and $G' = E'[x := \beta E']$. Now we have $d \Rightarrow d'$ and $G \Rightarrow G'$ by the induction hypothesis, hence $[\beta]E[G[\mu\gamma.d]] \Rightarrow d'[\gamma := \beta E'G']$. By the Barendregt convention we have $\gamma \neq \alpha$ and $\gamma \notin F$, so $\gamma \notin \mathrm{FV}(E')$ by Fact 4.3.6. Furthermore we have $c'[\gamma := \alpha E'][\alpha := \beta E'] \equiv c'[\alpha := \beta E'][\gamma := \beta E'G']$ by Lemma 4.2.12, so $([\alpha]F[\mu\gamma.c])[\alpha := \beta E] \Rightarrow c'[\gamma := \alpha E'][\alpha := \beta E']$. □

In order to define the complete development we are required to decide which redexes to contract. However, this job is non-trivial because $\Rightarrow$ is very strong. That is, in one step it is able to move a subterm that is located very deep into the term to the outside. For example, let us consider the command e.

$$e \equiv E_n[\mu\alpha_n.[\alpha_n] \dots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\alpha_0.c]] \dots]$$

For sake of simplicity, we suppose that $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $i, j \in \mathbb{N}$ such that $0 \leq j < i \leq n$, and moreover $\alpha_0 \notin \mathrm{FCV}(c)$. Intuitively one would be urged to contract the (t7)-redexes immediately, however, that does not yield the complete development. We have $[\alpha_{i+1}]E_i[\mu\alpha_i.d] \Rightarrow d'$ for each i such that $0 \leq i < n$, hence the whole command e reduces to c' (considering $c \Rightarrow c'$). As this example indicates, it is impossible to determine whether a (t7)-redex should be contracted without looking deeply into the term. In order to define the complete development we introduce the following case distinction on terms.

Lemma 4.3.9. *Given a term t, then t is of exactly one of the following shapes.*

variables
1. x

values
2. $\underline{n}$
3. $\lambda x.s$

redexes
4. $(\lambda x.s)r$
5. $\mathsf{nrec}\ r\ s\ \underline{n}$
6. $E_n[\mu\alpha_n.[\alpha_n] \dots E_1[\mu\alpha_1.[\alpha_1]r] \dots]$
   with $n \geq 1$, $\alpha_i \notin \mathrm{FCV}(r)$ for all $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ for all i and j such that $1 \leq j < i \leq n$, and $r \not\equiv E[\mu\beta.c]$
7. $E_n[\mu\alpha_n.[\alpha_n] \dots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.c]] \dots]$
   with $\alpha_i \notin \mathrm{FCV}(c)$ for all $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ for all i and j such that $0 \leq j < i \leq n$, and if $c \equiv [\beta]t$ then $\beta \in \mathrm{FCV}(t)$

other
8. sr
   with $s \not\equiv E[\mu\beta.c]$ and $s \not\equiv \lambda x.t$
9. $\mathsf{nrec}\ r\ s\ u$
   with $u \not\equiv E[\mu\beta.c]$ and $u \not\equiv \underline{n}$
10. $\mathsf{S}u$
    with $u \not\equiv E[\mu\beta.c]$ and $u \not\equiv \underline{n}$

Proof. We prove that t is always of one of the given shapes by induction on the structure of t. Furthermore, because these shapes are non-overlapping it is immediate that t is always of exactly one of these shapes. □

Lemma 4.3.10. *Given continuation variables $\alpha_1, \ldots \alpha_n$, contexts $E_1, \ldots, E_n$, $E'_1, \ldots, E'_n$ and terms r and r', such that:*

1. *$\alpha_i \notin \mathrm{FCV}(r)$ and $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $j < i$,*
2. *for every context F_i such that $E_i \Rightarrow F_i$ we have $F_i \Rightarrow E'_i$,*
3. *for every term s such that $r \Rightarrow s$ we have $s \Rightarrow r'$, and,*
4. *$r \not\equiv E[\mu\beta.d]$,*

then:

1. *$E_n[\mu\alpha_n.[\alpha_n] \ldots E_1[\mu\alpha_1.[\alpha_1]r]] \Rightarrow t$ implies $t \Rightarrow E'_n \ldots E'_1[r']$, and,*
2. *$[\alpha]E_n[\mu\alpha_n.[\alpha_n] \ldots E_1[\mu\alpha_1.[\alpha_1]r]] \Rightarrow c$ implies $c \Rightarrow [\alpha]E'_n \ldots E'_1[r']$.*

Proof. To prove these properties we have to strengthen the second property:

2. $[\alpha]E_n[\mu\alpha_n.[\alpha_n] \ldots E_1[\mu\alpha_1.[\alpha_1]r]] \Rightarrow c$ implies $c \equiv [\alpha]v$ and $v \Rightarrow E'_n \ldots E'_1[r']$. Also, $\alpha \notin \mathrm{FCV}(r)$ and $\alpha \notin \mathrm{FCV}(E_i)$ for all i implies $\alpha \notin \mathrm{FCV}(v)$.

We prove these properties simultaneously by induction on n.

1. Suppose that $n = 0$. Now the first property holds by assumption. For the second property, the only possible reduction on $[\alpha]r$ is (c1) because $r \not\equiv E[\mu\beta.d]$. Therefore we have $c \equiv [\alpha]s$ and $r \Rightarrow s$. Now we are done because we have $s \Rightarrow r'$ by assumption. Furthermore, if we suppose that $\alpha \notin \mathrm{FCV}(r)$, then $\alpha \notin \mathrm{FCV}(s)$ by Fact 4.3.6, as required.

2. Suppose that $n > 0$. Now let $u \equiv E_{n-1}[\mu\alpha_{n-1}.[\alpha_{n-1}] \ldots E_1[\mu\alpha_1.[\alpha_1]r]]$ and $u' \equiv E'_{n-1} \ldots E'_1[r']$. For the first property, the following reductions are possible.

   (t4,t7) $E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow F_n[v]$ with $E_n \Rightarrow F_n$ and $u \Rightarrow v$. Now we have $v \Rightarrow u'$ by the induction hypothesis, hence $F_n[v] \Rightarrow E'_n[u']$ by assumption.

   (t4,t6) $E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow F^l_n[\mu\alpha_n.c[\alpha_n := \alpha_n F^r_n]]$ with $E_n = E^l_n E^r_n$, $E^l_n \Rightarrow F^l_n$, $E^r_n \Rightarrow F^r_n$ and $[\alpha_n]u \Rightarrow c$. Now we have $c \equiv [\alpha_n]v$ and $v \Rightarrow u'$ by the induction hypothesis. Also, we have $\alpha_n \notin \mathrm{FCV}(u)$ by assumption and therefore $\alpha_n \notin \mathrm{FCV}(v)$ by the induction hypothesis. Hence we have the following.

$$\begin{aligned} F^l_n[\mu\alpha_n.c[\alpha_n := \alpha_n F^r_n]] &\equiv F^l_n[\mu\alpha_n.([\alpha_n]v)[\alpha_n := \alpha_n F^r_n]] \\ &\equiv F^l_n[\mu\alpha_n.[\alpha_n]F^r_n[v]] \\ &\Rightarrow E'^l_n E'^r_n[u'] \\ &\equiv E'_n[u'] \end{aligned}$$

   For the second property we have to consider the same reductions as above, but surrounded by a (c1)-reduction, and the following reduction.

(c2) $[\alpha]E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow c[\alpha_n := \alpha F_n]$ with $E_n \Rightarrow F_n$ and $[\alpha_n]u \Rightarrow c$. Now we have $c \equiv [\alpha_n]v$ and $v \Rightarrow u'$ by the induction hypothesis. Also, we have $\alpha_n \notin \mathrm{FCV}(u)$ by assumption and therefore $\alpha_n \notin \mathrm{FCV}(v)$ by the induction hypothesis. Hence we have the following.

$$c[\alpha_n := \alpha F_n] \equiv ([\alpha_n]v)[\alpha_n := \alpha F_n] \equiv [\alpha]F_n[v] \Rightarrow [\alpha]E'_n[u']$$

Furthermore, let us suppose that $\alpha \notin \mathrm{FCV}(E_n) \cup \mathrm{FCV}(u)$, then $\alpha \notin \mathrm{FCV}(F_n) \cup \mathrm{FCV}(v)$ by Fact 4.3.6, so $\alpha \notin \mathrm{FCV}(F_n[v])$. □

Lemma 4.3.11. *Given continuation variables $\alpha_1, \ldots \alpha_n$, contexts $E_0, \ldots, E_n$, $E'_0, \ldots, E'_n$ and commands d and d', such that:*

1. *$\alpha_i \notin \mathrm{FCV}(d)$ and $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $j < i$,*
2. *for every context F_i such that $E_i \Rightarrow F_i$ we have $F_i \Rightarrow E'_i$,*
3. *for every command e such that $d \Rightarrow e$ we have $e \Rightarrow d'$, and,*
4. *$d \equiv [\alpha]t$ implies $\alpha \in \mathrm{FCV}(t)$,*

then:

1. *$E_n[\mu\alpha_n.[\alpha_n] \ldots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.d]]] \Rightarrow t$ implies $t \Rightarrow \mu\alpha.d'[\beta := \alpha E'_n \ldots E'_0]$*
2. *$[\alpha]E_n[\mu\alpha_n.[\alpha_n] \ldots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.d]]] \Rightarrow c$ implies $c \Rightarrow d'[\beta := \alpha E'_n \ldots E'_0]$*

Proof. To prove these properties we have to strengthen the first property:

1. $E_n[\mu\alpha_n.[\alpha_n] \ldots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.d]]] \Rightarrow t$ and moreover $F \Rightarrow E'$ implies $F[t] \Rightarrow \mu\alpha.d'[\beta := \alpha E' E'_n \ldots E'_0]$ and $[\alpha]F[t] \Rightarrow d'[\beta := \alpha E' E'_n \ldots E'_0]$.

We prove these properties simultaneously by induction on n.

1. Suppose that $n = 0$. For the first property, merely the following reduction is possible.

(t4,t6) $E_0[\mu\beta.d] \Rightarrow F^l_0[\mu\beta.e[\beta := \beta F^r_0]]$ such that $E_0 = E^l_0 E^r_0$, $E^l_0 \Rightarrow F^l_0$, $E^r_0 \Rightarrow F^r_0$ and $d \Rightarrow e$. Now we have the following.

$$\begin{aligned} F F^l_0[\mu\beta.e[\beta := \beta F^r_0]] &\Rightarrow \mu\beta.d'[\beta := \beta E'^r_0][\beta := \beta E' E'^l_0] \\ &\equiv \mu\alpha.d'[\beta := \alpha E' E'_0] \\ [\alpha]F F^l_0[\mu\beta.e[\beta := \beta F^r_0]] &\Rightarrow d'[\beta := \beta E'^r_0][\beta := \alpha E' E'^l_0] \\ &\equiv d'[\beta := \alpha E' E'_0] \end{aligned}$$

For the second property, the following reductions are possible.

(c1,t4,t6) $[\alpha]E_0[\mu\beta.d] \Rightarrow [\alpha]F^l_0[\mu\beta.e[\beta := \beta F^r_0]]$ such that $E_0 = E^l_0 E^r_0$, $E^l_0 \Rightarrow F^l_0$, $E^r_0 \Rightarrow F^r_0$ and $d \Rightarrow e$. Now we have $e[\beta := \alpha F^r_0] \Rightarrow d'[\beta := \alpha E'^r_0]$ by Lemma 4.3.8 and therefore the following.

$$[\alpha]F^l_0[\mu\beta.e[\beta := \beta F^r_0]] \Rightarrow d'[\beta := \beta E'^r_0][\beta := \alpha E'^l_0] \equiv d'[\beta := \alpha E'_0]$$

(c2) $[\alpha]E_0[\mu\beta.d] \Rightarrow e[\beta := \alpha F_0]$ such that $E_0 \Rightarrow F_0$ and $d \Rightarrow e$. Now we have $e[\beta := \alpha F_0] \Rightarrow d'[\beta := \alpha E'_0]$ by Lemma 4.3.8.

2. Suppose that $n > 0$. Let $u \equiv E_{n-1}[\mu\alpha_{n-1}.[\alpha_{n-1}] \dots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.c]]]$. For the first property, the following reductions are possible.

(t4,t7) $E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow F_n[v]$ such that $E_n \Rightarrow F_n$ and $u \Rightarrow v$. Now we have, by the induction hypothesis, $FF_n[v] \Rightarrow \mu\alpha.d'[\alpha := \alpha E' E'_n \dots E'_0]$ and $[\alpha]FF_n[v] \Rightarrow d'[\alpha := \alpha E' E'_n \dots E'_0]$.

(t4,t6) $E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow F^l_n[\mu\alpha_n.e[\alpha_n := \alpha_n F^r_n]]$ such that $E_n = E^l_n E^r_n$, $E^l_n \Rightarrow F^l_n$, $E^r_n \Rightarrow F^r_n$ and $[\alpha_n]u \Rightarrow e$. Now we have $e \Rightarrow f'$ where $f' \equiv d'[\beta := \alpha_n E'_{n-1} \dots E'_0]$ by the induction hypothesis. Furthermore we have $e[\alpha_n := \alpha_n F^r_n] \Rightarrow f'[\alpha_n := \alpha_n E'^r_n]$ by Lemma 4.3.8 and therefore the following.

$$\begin{aligned} FF^l_n[\mu\alpha_n.e[\alpha_n := \alpha_n F^r_n]] &\Rightarrow \mu\alpha_n.f'[\alpha_n := \alpha_n E'^r_n][\alpha_n := \alpha_n E' E'^l_n] \\ &\equiv \mu\alpha.d'[\beta := \alpha E' E'_n \dots E'_0] \\ [\alpha]FF^l_n[\mu\alpha_n.e[\alpha_n := \alpha_n F^r_n]] &\Rightarrow f'[\alpha_n := \alpha_n E'^r_n][\alpha_n := \alpha_n E' E'^l_n] \\ &\equiv d'[\beta := \alpha E' E'_n \dots E'_0] \end{aligned}$$

For the second property, the following reductions are possible.

(c1,t4,t7) $[\alpha]E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow [\alpha]F_n[v]$ with $E_n \Rightarrow F_n$ and $u \Rightarrow v$. Now we have $[\alpha]F_n[v] \Rightarrow d'[\alpha := \alpha E'_n \dots E'_0]$ by the induction hypothesis.

(c1,t4,t6) $[\alpha]E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow [\alpha]F^l_n[\mu\alpha_n.c[\alpha_n := \alpha_n F^r_n]]$ with $E_n = E^l_n E^r_n$, $E^l_n \Rightarrow F^l_n$, $E^r_n \Rightarrow F^r_n$ and $[\alpha_n]u \Rightarrow c$. Now we have $e \Rightarrow f'$ where $f' \equiv d'[\beta := \alpha_n E'_{n-1} \dots E'_0]$ by the induction hypothesis. Furthermore we have $e[\alpha_n := \alpha_n F^r_n] \Rightarrow f'[\alpha_n := \alpha_n E'^r_n]$ by Lemma 4.3.8 and therefore the following.

$$\begin{aligned} [\alpha]F^l_n[\mu\alpha_n.e[\alpha_n := \alpha_n F^r_n]] &\Rightarrow f'[\alpha_n := \alpha_n E'^r_n][\alpha_n := \alpha_n E'^l_n] \\ &\equiv d'[\beta := \alpha E'_n \dots E'_0] \end{aligned}$$

(c2) $[\alpha]E_n[\mu\alpha_n.[\alpha_n]u] \Rightarrow e[\alpha_n := \alpha F_n]$ with $E_n \Rightarrow F_n$ and $[\alpha_n]u \Rightarrow e$. Now we have $e \Rightarrow f'$ where $f' \equiv d'[\beta := \alpha_n E'_{n-1} \dots E'_0]$ by the induction hypothesis. Hence $e[\alpha_n := \alpha F_n] \Rightarrow f'[\alpha_n := \alpha E'_n]$ by Lemma 4.3.8, so $e[\alpha_n := \alpha F_n] \Rightarrow d'[\beta := \alpha_n E'_n \dots E'_0]$. □

Definition 4.3.12. *The* complete development $t^\diamond$ *of a term* t *is defined (using the case distinction made in Lemma 4.3.9) as:*

1. $x^\diamond := x$
2. $0^\diamond := 0$
3. $(\lambda x.s)^\diamond := \lambda x.s^\diamond$
4. $((\lambda x.s)r)^\diamond := s^\diamond[x := r^\diamond]$
5. $(\mathsf{nrec}\ r\ s\ 0)^\diamond := r^\diamond$
6. $(\mathsf{nrec}\ r\ s\ (\mathtt{S}\underline{n}))^\diamond := s^\diamond\ \underline{n}\ (\mathtt{nrec}\ r^\diamond\ s^\diamond\ \underline{n})$

7. $(E_n[\mu\alpha_n.[\alpha_n]\dots E_1[\mu\alpha_1.[\alpha_1]r]])^\diamond := E_n^\diamond \dots E_1^\diamond[r^\diamond]$
   provided that $n \geq 1$, $\alpha_i \notin \mathrm{FCV}(r)$ *for all* $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ *for all* i *and* j *such that* $0 \leq j < i \leq n$, *and* $r \not\equiv E[\mu\beta.c]$

8. $(E_n[\mu\alpha_n.[\alpha_n]\dots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.c]]])^\diamond := \mu\alpha.c^\diamond[\beta := \alpha E_n^\diamond \dots E_0^\diamond]$
   provided that $\alpha_i \notin \mathrm{FCV}(c)$ *for all* $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ *for all* i *and* j *such that* $0 \leq j < i \leq n$, *and if* $c \equiv [\beta]t$ *then* $\beta \in \mathrm{FCV}(t)$

9. $(sr)^\diamond := s^\diamond r^\diamond$
   provided that $s \not\equiv E[\mu\beta.c]$ *and* $s \not\equiv \lambda x.t$

10. $(\mathsf{nrec}\ r\ s\ u)^\diamond := \mathsf{nrec}\ r^\diamond\ s^\diamond\ u^\diamond$
    provided that $u \not\equiv E[\mu\beta.c]$ *and* $u \not\equiv \underline{n}$

11. $(\mathsf{S}u)^\diamond := \mathsf{S}u^\diamond$
    provided that $u \not\equiv E[\mu\beta.c]$

with the complete development $c^\diamond$ *of a command* c *defined as:*

1. $([\alpha]E[\mu\beta.c])^\diamond := c^\diamond[\beta := \alpha E^\diamond]$
2. $([\alpha]t)^\diamond := [\alpha]t^\diamond$
   provided that $t \not\equiv E[\mu\beta.c]$

and the complete development $E^\diamond$ *of a context* E *defined as:*

1. $\Box^\diamond := \Box$
2. $(Et)^\diamond := E^\diamond t^\diamond$
3. $(\mathsf{S}E)^\diamond := \mathsf{S}E^\diamond$
4. $(\mathsf{nrec}\ r\ s\ E)^\diamond := \mathsf{nrec}\ r^\diamond\ s^\diamond\ E^\diamond$

Theorem 4.3.13. *Given terms* t_1 *and* t_2 *such that* $t_1 \Rightarrow t_2$, *then* $t_2 \Rightarrow t_1^\diamond$.

Proof. We prove this result by mutual induction on the structure of terms, commands and contexts. Firstly, we prove it for terms using the case distinction made in Lemma 4.3.9.

1. Suppose that $t_1 \equiv x$. Now merely the reduction (t1) is possible, so let $x \Rightarrow x$. Therefore $x \Rightarrow x^\diamond \equiv x$.

2. Suppose that $t_1 \equiv 0$. Now merely the reduction (t2) is possible, so let $0 \Rightarrow 0$. Therefore $0 \Rightarrow 0^\diamond \equiv 0$.

3. Suppose that $t_1 \equiv \lambda x.s_1$. Now merely the reduction (t3) is possible, so let $\lambda x.s_1 \Rightarrow \lambda x.s_2$ with $s_1 \Rightarrow s_2$. Therefore $s_2 \Rightarrow s_1^\diamond$ by the induction hypothesis, so $\lambda x.s_2 \Rightarrow (\lambda x.s_1)^\diamond \equiv \lambda x.s_1^\diamond$.

4. Suppose that $t_1 \equiv (\lambda x.s_1)r_1$. The following reductions are possible.

(t4,t3) $(\lambda x.s_1)r_1 \Rightarrow (\lambda x.s_2)r_2$ with $s_1 \Rightarrow s_2$ and $r_1 \Rightarrow r_2$. Now we have $s_2 \Rightarrow s_1^\diamond$ and $r_2 \Rightarrow r_1^\diamond$ by the induction hypothesis. Therefore we have $(\lambda x.s_2)r_2 \Rightarrow ((\lambda x.s_1)r_1)^\diamond \equiv s_1^\diamond[x := r_1^\diamond]$.

(t5) $(\lambda x.s_1)r_1 \Rightarrow s_2[x := r_2]$ with $s_1 \Rightarrow s_2$ and $r_1 \Rightarrow r_2$. Now we have $s_2 \Rightarrow s_1^\diamond$ and $r_2 \Rightarrow r_1^\diamond$ by the induction hypothesis. Therefore $s_2[x := r_2] \Rightarrow ((\lambda x.s_1)r_1)^\diamond \equiv s_1^\diamond[x := r_1^\diamond]$ by Lemma 4.3.7.

5. Suppose that $t_1 \equiv \mathtt{nrec}\ r_1\ s_1\ 0$. The following reductions are possible.

(t4,t2) $\mathtt{nrec}\ r_1\ s_1\ 0 \Rightarrow \mathtt{nrec}\ r_2\ s_2\ 0$ with $r_1 \Rightarrow r_2$ and $s_1 \Rightarrow s_2$. Now we have $r_2 \Rightarrow r_1^\diamond$ and $s_2 \Rightarrow s_1^\diamond$ by the induction hypothesis, hence $\mathtt{nrec}\ r_2\ s_2\ 0 \Rightarrow (\mathtt{nrec}\ r_1\ s_2\ 0)^\diamond \equiv r_1^\diamond$.

(t8) $\mathtt{nrec}\ r_1\ s_1\ 0 \Rightarrow r_2$ with $r_1 \Rightarrow r_2$. Now we have $r_2 \Rightarrow r_1^\diamond$ by the induction hypothesis, hence $r_2 \Rightarrow (\mathtt{nrec}\ r_1\ s_2\ 0)^\diamond \equiv r_1^\diamond$.

6. Suppose that $t_1 \equiv \mathtt{nrec}\ r_1\ s_1\ (\mathtt{S}\underline{n})$. The following reductions are possible.

(t4,t2) $\mathtt{nrec}\ r_1\ s_1\ (\mathtt{S}\underline{n}) \Rightarrow \mathtt{nrec}\ r_2\ s_2\ (\mathtt{S}\underline{n})$, then $r_1 \Rightarrow r_2$ and $s_1 \Rightarrow s_2$. Now we have $r_2 \Rightarrow r_1^\diamond$ and $s_2 \Rightarrow s_1^\diamond$ by the induction hypothesis, hence $\mathtt{nrec}\ r_2\ s_2\ (\mathtt{S}\underline{n}) \Rightarrow (\mathtt{nrec}\ r_1\ s_2\ (\mathtt{S}\underline{n}))^\diamond \equiv s_1^\diamond\ \underline{n}\ (\mathtt{nrec}\ r_1^\diamond\ s_1^\diamond\ \underline{n})$.

(t9) $\mathtt{nrec}\ r_1\ s_1\ (\mathtt{S}\underline{n}) \Rightarrow s_2\ \underline{n}\ (\mathtt{nrec}\ r_2\ s_2\ \underline{n})$, then $r_1 \Rightarrow r_2$ and $s_1 \Rightarrow s_2$. Now we have $r_2 \Rightarrow r_1^\diamond$ and $s_2 \Rightarrow s_1^\diamond$ by the induction hypothesis, hence $s_2\ \underline{n}\ (\mathtt{nrec}\ r_2\ s_2\ \underline{n}) \Rightarrow (\mathtt{nrec}\ r_1\ s_2\ (\mathtt{S}\underline{n}))^\diamond \equiv s_1^\diamond\ \underline{n}\ (\mathtt{nrec}\ r_1^\diamond\ s_1^\diamond\ \underline{n})$.

7. Suppose that $t_1 \equiv E_n[\mu\alpha_n.[\alpha_n] \dots E_1[\mu\alpha_1.[\alpha_1]r]]$ provided that $n \geq 1$, $\alpha_i \notin \mathrm{FCV}(r)$ for all $i \in \mathbb{N}$ such that $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $i, j \in \mathbb{N}$ such that $0 \leq j < i \leq n$, and $r \not\equiv E[\mu\beta.c]$. Now we have $E_i \Rightarrow F_i$ implies $F_i \Rightarrow F_i^\diamond$ and $r \Rightarrow s$ implies $s \Rightarrow r^\diamond$ by the induction hypothesis. Hence if $t_1 \Rightarrow t_2$, then $t_2 \Rightarrow t_1^\diamond \equiv E_n^\diamond \dots E_1^\diamond[r^\diamond]$ by Lemma 4.3.10.

8. Suppose that $t_1 \equiv E_n[\mu\alpha_n.[\alpha_n] \dots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.c]]]$ provided that $\alpha_i \notin \mathrm{FCV}(c)$ for all $i \in \mathbb{N}$ such that $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $i, j \in \mathbb{N}$ such that $0 \leq j < i \leq n$, and if $c \equiv [\beta]t$ then $\beta \in \mathrm{FCV}(t)$. Now we have $E_i \Rightarrow F_i$ implies $F_i \Rightarrow F_i^\diamond$ and $c \Rightarrow d$ implies $d \Rightarrow c^\diamond$ by the induction hypothesis. Hence if $t_1 \Rightarrow t_2$, then $t_2 \Rightarrow t_1^\diamond \equiv \mu\alpha.c^\diamond[\beta := \alpha E_n^\diamond \dots E_0^\diamond]$ by Lemma 4.3.11.

9. Suppose that $t_1 \equiv s_1 r_1$ provided that $s_1 \neq E[\mu\alpha.c]$ and $s_1 \neq \lambda x.s$. Now merely the reduction (t4) is possible, therefore let $s_1 r_1 \Rightarrow s_2 r_2$ with $s_1 \Rightarrow s_2$ and $r_1 \Rightarrow r_2$. We have $s_1 \Rightarrow s_2^\diamond$ and $r_2 \Rightarrow r_1^\diamond$ by the induction hypothesis, so $s_2 r_2 \Rightarrow (s_1 r_1)^\diamond \equiv s_1^\diamond r_1^\diamond$.

10. Suppose that $t_1 \equiv \mathtt{nrec}\ r_1\ s_1\ u_1$ provided that $u_1 \neq E[\mu\alpha.c]$ and $u_1 \neq \underline{n}$. Now we have $\mathtt{nrec}\ r_1\ s_1\ u_1 \Rightarrow \mathtt{nrec}\ r_2\ s_2\ u_2$ with $r_1 \Rightarrow r_2$, $s_1 \Rightarrow s_2$ and $u_1 \Rightarrow u_2$ because merely the reduction (t4) is possible. Furthermore we have $r_2 \Rightarrow r_1^\diamond$, $s_2 \Rightarrow s_1^\diamond$ and $u_2 \Rightarrow u_1^\diamond$ by the induction hypothesis, so $\mathtt{nrec}\ r_2\ s_2\ u_2 \Rightarrow (\mathtt{nrec}\ r_1\ s_1\ u_1)^\diamond \equiv \mathtt{nrec}\ r_1^\diamond\ s_1^\diamond\ u_1^\diamond$.

11. Suppose that $t_1 \equiv \mathtt{S}u_1$ provided that $u_1 \neq E[\mu\alpha.c]$. Now merely the reduction (t4) is possible, therefore let $\mathtt{S}u_1 \Rightarrow \mathtt{S}u_2$ with $u_1 \Rightarrow u_2$. We have $u_2 \Rightarrow u_1^\diamond$ by the induction hypothesis, so $\mathtt{S}u_2 \Rightarrow (\mathtt{S}u_1)^\diamond \equiv \mathtt{S}u_1^\diamond$.

Secondly, we prove it for commands. Here we distinguish the following cases.

1. Suppose that $c_1 \equiv [\alpha]E_n[\mu\alpha_n.[\alpha_n]\dots E_1[\mu\alpha_1.[\alpha_1]r]]$ provided that $\alpha_i \notin \mathrm{FCV}(r)$ for all $i \in \mathbb{N}$ such that $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $i, j \in \mathbb{N}$ such that $0 \leq j < i \leq n$, and $r \not\equiv E[\mu\beta.c]$. Now we have $E_i \Rightarrow F_i$ implies $F_i \Rightarrow F_i^\diamond$ and $r \Rightarrow s$ implies $s \Rightarrow r^\diamond$ by the induction hypothesis. Hence if $c_1 \Rightarrow c_2$, we have $c_2 \Rightarrow c_1^\diamond \equiv [\alpha]E_n^\diamond \dots E_1^\diamond[r^\diamond]$ by Lemma 4.3.10.

2. Suppose that $c_1 \equiv [\alpha]E_n[\mu\alpha_n.[\alpha_n]\dots E_1[\mu\alpha_1.[\alpha_1]E_0[\mu\beta.c]]]$ provided that $\alpha_i \notin \mathrm{FCV}(c)$ for all $i \in \mathbb{N}$ such that $1 \leq i \leq n$, $\alpha_i \notin \mathrm{FCV}(E_j)$ for all $i, j \in \mathbb{N}$ such that $0 \leq j < i \leq n$, and if $c \equiv [\beta]t$ then $\beta \in \mathrm{FCV}(t)$. Now we have $E_i \Rightarrow F_i$ implies $F_i \Rightarrow F_i^\diamond$ and $c \Rightarrow d$ implies $d \Rightarrow c^\diamond$ by the induction hypothesis. Hence if $c_1 \Rightarrow c_2$, then $c_2 \Rightarrow c_1^\diamond \equiv c^\diamond[\beta := \alpha E_n^\diamond \dots E_0^\diamond]$ by Lemma 4.3.11.

Thirdly, for contexts it follows immediately, so we are done. □

Corollary 4.3.14. *Parallel reduction is confluent, that is, if $t_1 \Rightarrow t_2$ and $t_1 \Rightarrow t_3$, then there exists a term t_4 such that $t_2 \Rightarrow t_4$ and $t_3 \Rightarrow t_4$.*

Proof. Let $t_4 = t_1^\diamond$. Now we have $t_2 \Rightarrow t_1^\diamond$ and $t_3 \Rightarrow t_1^\diamond$ by Theorem 4.3.13. □

Lemma 4.3.15. *Given terms t and t' such that $t \to t'$, then $t \Rightarrow t'$.*

Proof. By induction on $t \to t'$ using the fact that parallel reduction is reflexive (Lemma 4.3.3). □

Lemma 4.3.16. *Given terms t, t' and s such that $t \to t'$, then:*

$$t[x := s] \to t'[x := s]$$

Proof. By induction on $t \to t'$ using Lemma 4.2.12. □

Lemma 4.3.17. *Given terms t, s and s' such that $s \twoheadrightarrow s'$, then:*

$$t[x := s] \twoheadrightarrow t[x := s']$$

Proof. By induction on the structure of t using Lemma 4.2.12. □

Corollary 4.3.18. *Reduction is preserved under substitution, that is given terms t and t' such that $t \twoheadrightarrow t'$ and terms s and s' such that $s \twoheadrightarrow s'$, then:*

$$t[x := s] \twoheadrightarrow t'[x := s']$$

Proof. By induction on $t \twoheadrightarrow t'$.

1. Let $t \equiv t'$. Now $t[x := s] \equiv t'[x := s] \twoheadrightarrow t'[x := s']$ by Lemma 4.3.16.

2. Let $t \to t'$. Now $t[x := s] \to t'[x := s] \twoheadrightarrow t'[x := s']$ by Lemma 4.3.17 and 4.3.16, respectively.

3. Let $t \twoheadrightarrow t'$ and $t' \twoheadrightarrow t''$. Now $t[x := s] \twoheadrightarrow t'[x := s] \twoheadrightarrow t''[x := s']$ by the induction hypothesis. □

Definition 4.3.19. *Reduction $E \to E'$ on contexts E and E' is defined as the compatible closure of $\to$ on contexts. Moreover, $\twoheadrightarrow$ on contexts denotes the reflexive/transitive closure of $\to$ on contexts.*

Lemma 4.3.20. *Given terms t and t' and a context E such that $t \to t'$, then:*

$$t[\alpha := \beta E] \twoheadrightarrow t'[\alpha := \beta E]$$

Proof. By induction on $t \to t'$ using Lemma 4.2.12. The only interesting case is $[\alpha]\mu\gamma.c \to c[\gamma := \alpha\Box]$, where we have the following.

$$\begin{aligned}
([\alpha]\mu\gamma.c)[\alpha := \beta E] &\equiv [\beta]E[\mu\gamma.c[\alpha := \beta E]] \\
&\twoheadrightarrow [\beta]\mu\gamma.c[\alpha := \beta E][\gamma := \gamma E] && \text{(a)} \\
&\to c[\alpha := \beta E][\gamma := \gamma E][\gamma := \beta\Box] \\
&\equiv c[\gamma := \alpha\Box][\alpha := \beta E] && \text{(b)}
\end{aligned}$$

Here we use Lemma 4.2.9 for (a) and Lemma 4.2.12 for (b). □

Lemma 4.3.21. *Given a term t and contexts E and E' such that $E \to E'$, then:*

$$t[\alpha := \beta E] \twoheadrightarrow t[\alpha := \beta E']$$

Proof. By induction on the structure of t using Lemma 4.2.12. □

Corollary 4.3.22. *Reduction is preserved under structural substitution, that is given terms t and t' such that $t \twoheadrightarrow t'$ and contexts E and E' such that $E \twoheadrightarrow E'$, then:*

$$t[\alpha := \beta E] \twoheadrightarrow t'[\alpha := \beta E']$$

Proof. Similar to the proof of Corollary 4.3.18. □

Lemma 4.3.23. *Given terms t and t' such that $t \Rightarrow t'$, then $t \twoheadrightarrow t'$.*

Proof. By induction on $t \Rightarrow t'$. We treat some interesting cases.

(t5) Let $(\lambda x.t)r \Rightarrow t'[x := r']$ with $t \Rightarrow t'$ and $r \Rightarrow r'$. We have $t \twoheadrightarrow t'$ and $r \twoheadrightarrow r'$ by the induction hypothesis. Therefore, by Corollary 4.3.18, we have $(\lambda x.t)r \to t[x := r] \twoheadrightarrow t'[x := r']$.

(t6) Let $E[\mu\alpha.c] \Rightarrow \mu\alpha.c'[\alpha := \alpha E']$ with $c \Rightarrow c'$ and $E \Rightarrow E'$. We have $c \twoheadrightarrow c'$ and $E \twoheadrightarrow E'$ by the induction hypothesis. Therefore, by Lemma 4.2.9 and Corollary 4.3.22: $E[\mu\alpha.c] \twoheadrightarrow \mu\alpha.c[\alpha := \alpha E] \twoheadrightarrow \mu\alpha.c'[\alpha := \alpha E']$. □

Theorem 4.3.24. *Reduction on $\lambda_\mu^{\mathbf{T}}$ is confluent, that is if $t_1 \twoheadrightarrow t_2$ and $t_1 \twoheadrightarrow t_3$, then there exists a term t_4 such that $t_2 \twoheadrightarrow t_4$ and $t_3 \twoheadrightarrow t_4$.*

Proof. By Corollary 4.3.14, Lemma 4.3.15, Lemma 4.3.23 and the fact that if $\Rightarrow$ is confluent then also $\Rightarrow^*$ is confluent. □

4.4 Strong normalization of $\lambda_\mu^{\mathbf{T}}$

In this section we prove that $\lambda_\mu^{\mathbf{T}}$ is strongly normalizing. This is achieved by defining two reductions $\to_A$ and $\to_B$ such that $\to = \to_A \cup \to_B$. In Section 4.4.1 we prove, using the reducibility method, that $\to_A$ is strongly normalizing. In Section 4.4.2 we prove that $\to_B$ is strongly normalizing and moreover that each infinite $\to_{AB}$-reduction sequence can be transformed into an infinite $\to_A$-reduction sequence. The first phase is inspired by Parigot's proof of strong normalization for λ_μ [Par97] and the second phase is inspired by Rehof and Sørensen's proof of strong normalization for λ_Δ [RS94].

Definition 4.4.1. *Let $\to_A$ denote the compatible closure of the reduction rules $\to_\beta$, $\to_{\mu\mathtt{S}}$, $\to_{\mu R}$, $\to_0$, $\to_{\mathtt{S}}$ and $\to_{\mu\mathbb{N}}$, moreover let $\to_B$ denote the compatible closure of the reduction rules $\to_{\mu\eta}$ and $\to_{\mu i}$.*

Definition 4.4.2. *Given a notion of reduction $\to_X$ (e.g. $\to_A$ or $\to_B$), let $\mathtt{SN}_X$ and $\mathtt{SN}_X^\perp$ denote the sets of* strongly normalizing terms *and* strongly normalizing commands*, respectively.*

Definition 4.4.3. *Given a notion of reduction $\to_X$ (e.g. $\to_A$ or $\to_B$) and a term $t \in \mathtt{SN}_X$ or command $c \in \mathtt{SN}_X^\perp$, let $\nu_X(t)$ and $\nu_X(c)$ denote the* maximal length *of any $\to_X$-reduction sequence starting at t and c, respectively.*

4.4.1 Strong normalization of (A)

In this subsection we prove that $\to_A$-reduction is strongly normalization using the reducibility method. Since we are merely concerned with $\to_A$-reduction we will omit subscripts from all notations. Moreover, for the same reasons as in the previous section we specify most of the forthcoming lemmas just for terms and not for commands.

The reducibility method is originally due to Tait [Tai67], who proposed the following interpretation for $\to$-types.

$$\begin{aligned} [\![\alpha]\!] &:= \mathtt{SN} \\ [\![\rho \to \delta]\!] &:= \{t \mid \forall r \in [\![\rho]\!] \,.\, tr \in [\![\delta]\!]\} \end{aligned}$$

This interpretation makes it possible to prove strong normalization of $\lambda\to$ in a very short and elegant way [Geu08, for example]. Instead of proving that a term t of type ρ is strongly normalizing one proves a slight generalization, namely $t \in [\![\rho]\!]$. This method also extends easily to $\lambda^{\mathbf{T}}$ [GTL89, for example].

Unfortunately, for λ_μ it becomes more complicated. If a term of the shape $\lambda x.r$ consumes an argument the λ-abstraction vanishes. However, if a term of the shape $\mu\alpha.c$ consumes an argument the μ-abstraction remains, hence it is not possible to predict how many arguments $\mu\alpha.c$ will consume. To repair this issue Parigot has proposed a way to switch between a term that is a member of a certain *reducibility candidate* and that is strongly normalizing when applied to a certain set of sequences of arguments.

In $\lambda_\mu^{\mathbf{T}}$ a term of the shape $\mu\alpha.c$ is not only able to consume arguments on its right hand side, but is also able to consume an unknown number of $\mathtt{S}$'s and $\mathtt{nrec}$'s. Therefore we generalize Parigot's idea to contexts so that we are able

to switch between a term that is a member of a certain reducibility candidate and that is strongly normalizing in a certain set of contexts.

Before going into the details of the proof we state a fact and two lemmas about preservation of reduction under substitution.

Fact 4.4.4. *If $t \in \mathtt{SN}$ and $t \to t'$, then $\nu(t) > \nu(t')$.*

Lemma 4.4.5. *(A)-reduction is preserved under substitution, that is given terms t, t' and s, s' such that $t \twoheadrightarrow t'$ and $s \twoheadrightarrow s'$, then:*

$$t[x := r] \twoheadrightarrow t'[x := r]$$

Proof. Similar to the proof of Corollary 4.3.18. □

Lemma 4.4.6. *(A)-reduction is preserved under structural substitution, that is given terms t and t' such that $t \twoheadrightarrow t'$ and contexts E and E' such that $E \twoheadrightarrow E'$, then:*

$$t[\alpha := \beta E] \twoheadrightarrow t'[\alpha := \beta E']$$

Proof. Similar to the proof of Corollary 4.3.22. □

We now extend the notion of strongly normalizing terms to strongly normalizing contexts, informally this means that all terms contained in a context are strongly normalizing.

Definition 4.4.7. *Let $\mathtt{SN}^\Box$ denote the set of* strongly normalizing contexts*, this set is inductively defined as follows.*

1. $\Box \in \mathtt{SN}^\Box$
2. *If $E \in \mathtt{SN}^\Box$ and $t \in \mathtt{SN}$, then $Et \in \mathtt{SN}^\Box$.*
3. *If $E \in \mathtt{SN}^\Box$, then $\mathtt{S}E \in \mathtt{SN}^\Box$.*
4. *If $E \in \mathtt{SN}^\Box$, $r \in \mathtt{SN}$ and $s \in \mathtt{SN}$, then $\mathtt{nrec}\ r\ s\ E \in \mathtt{SN}^\Box$.*

Parigot's approach has another advantage; for the expansion lemmas we do not need to worry about the interpretation of types. We merely need the notion of being strongly normalizing (with respect to some context).

Lemma 4.4.8. *For each context E and redex w such that $E[w] \to t$ we have:*

1. *$t \equiv E[w']$ and $w \to w'$, or,*
2. *$t \equiv E'[w]$ and $E \to E'$*

A redex is a term w such that $w \equiv (\lambda x.r)t$, $w \equiv \mathtt{nrec}\ r\ s\ \underline{n}$ or $w \equiv E^s[\mu\alpha.c]$.

Proof. Simultaneous with $E[w] \not\equiv \lambda y.s$, $E[w] \not\equiv \mu\beta.d$ and $E[w] \not\equiv \underline{m}$ by induction on the structure of E. □

Lemma 4.4.9. *For each context E and terms t and $r \in \mathtt{SN}$ we have that $E[t[x := r]] \in \mathtt{SN}$ implies $E[(\lambda x.t)r] \in \mathtt{SN}$.*

Proof. We prove this by well-founded induction on $\nu(r) + \nu(E[t[x := r]])$. Since it is sufficient to show that $E[(\lambda x.t)r] \to w$ implies $w \in \mathtt{SN}$ we take a look at all possible shapes of w.

1. Suppose that $w \equiv E[t[x := r]]$. Now we are immediately done because $E[t[x := r]] \in \mathtt{SN}$ by assumption.

2. Suppose that $w \equiv E[(\lambda x.t')r]$ and $t \to t'$. Now $E[t[x := r]] \to E[t'[x := r]]$ by Lemma 4.4.5, hence $E[t'[x := r]] \in \mathtt{SN}$. Furthermore, by Fact 4.4.4, we have $\nu(E[t[x := r]]) > \nu(E[t'[x := r]])$, so we are allowed to use the induction hypothesis by which we have $E[(\lambda x.t')r] \in \mathtt{SN}$.

3. Suppose that $w \equiv E[(\lambda x.t)r']$ and $r \to r'$. Now $E[t[x := r]] \twoheadrightarrow E[t[x := r']]$ by Lemma 4.4.5 and therefore $E[t[x := r']] \in \mathtt{SN}$. Furthermore, by Fact 4.4.4, we have $\nu(r) > \nu(r')$, so we are allowed to use the induction hypothesis by which we have $E[(\lambda x.t)r'] \in \mathtt{SN}$.

4. Suppose that $w \equiv E[(\lambda x.t)r]$ and $E \to E'$. Now $E[t[x := r]] \to E'[t[x := r]]$ and therefore $E'[t[x := r]] \in \mathtt{SN}$. Furthermore, by Fact 4.4.4, we have $\nu(E[(\lambda x.t)r]) > \nu(E'[(\lambda x.t)r])$, so we are allowed to use the induction hypothesis by which we obtain $E'[(\lambda x.t)r] \in \mathtt{SN}$.

Lemma 4.4.8 guarantees that these are indeed all possible shapes of w. □

Lemma 4.4.10. *For each singular context $F^s \in \mathtt{SN}^\Box$, context E and command c we have that $E[\mu\alpha.c[\alpha := \alpha F^s]] \in \mathtt{SN}$ implies $E[F^s[\mu\alpha.c]] \in \mathtt{SN}$.*

Proof. By distinguishing the possible shapes on F^s we obtain three goals similar to the preceding lemma. Likewise, the proofs of these goals are similar to the proofs of the preceding lemma, only for the sub-cases corresponding to (1), (2) and (3) we use Lemma 4.4.6. □

Corollary 4.4.11. *For each context $F \in \mathtt{SN}^\Box$, context E and command c we have that $E[\mu\alpha.c[\alpha := \alpha F]] \in \mathtt{SN}$ implies $E[F[\mu\alpha.c]] \in \mathtt{SN}$.*

Proof. By induction on the structure of F.

1. Suppose that $F = \Box$. Now we have $E[\mu\alpha.c] \equiv E[\mu\alpha.c[\alpha := \alpha\Box]]$ for each context E and command c, so by assumption we are done.

2. Suppose that $F = G^s H$. Now, by Lemma 4.2.12 and assumption, we have $E[\mu\alpha.c[\alpha := \alpha H][\alpha := \alpha G^s]] \equiv E[\mu\alpha.c[\alpha := \alpha F]] \in \mathtt{SN}$. Therefore we have $E[G^s[\mu\alpha.c[\alpha := \alpha H]]] \in \mathtt{SN}$ by Lemma 4.4.10, so $E[G^s[H[\mu\alpha.c]]] \in \mathtt{SN}$ by the induction hypothesis. □

Lemma 4.4.12. *For each context E we have the following.*

1. *If $E[r] \in \mathtt{SN}$ and $s \in \mathtt{SN}$, then $E[\mathtt{nrec}\ r\ s\ 0] \in \mathtt{SN}$.*

2. *If $E[s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n})] \in \mathtt{SN}$, then $E[\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n})] \in \mathtt{SN}$.*

Proof. By well-founded induction on $\nu(E[\mathtt{nrec}\ r\ s\ 0])$ and $\nu(E[\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n})])$, respectively, we prove that if $E[\mathtt{nrec}\ r\ s\ 0] \to w$ or $E[\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n})] \to w$, then $w \in \mathtt{SN}$ by analyzing the possible shapes of w. □

Parigot's method extends the well-known *functional construction* of two sets S and T of terms $S \to T := \{t \mid \forall u \in S \;.\; tu \in T\}$ to a set $\mathcal{S}$ of sequences of terms and a set T of terms as $\mathcal{S} \to T := \{t \mid \forall \vec{u} \in \mathcal{S} \;.\; t\vec{u} \in T\}$. Moreover he has defined his notion of reducibility candidates in such way that each reducibility candidate R can be expressed as $\mathcal{S} \to \mathtt{SN}$ for a certain set of sequences of terms $\mathcal{S}$. Therefore he is able to switch between the proposition $t \in R$ and the proposition $t\vec{u} \in \mathtt{SN}$ for all $\vec{v} \in \mathcal{S}$. Since we consider contexts we extend Parigot's notion of functional construction to contexts, this happens in the obvious way.

Definition 4.4.13. *Given a set of contexts $\mathcal{E}$ and a set of terms T, then the* functional construction $X \to T$ *is defined as follows.*

$$\mathcal{E} \to T := \{t \mid \forall E \in \mathcal{E} \;.\; E[t] \in T\}$$

Given two sets of terms S and T, then $S \to T$ is defined as follows.

$$S \to T := \{\Box u \mid u \in S\} \to T$$

Remark that our definition of the functional construction $S \to T$ for sets of terms S and T is equivalent to the ordinary definition.

$$S \to T = \{\Box u \mid u \in S\} \to T = \{t \mid \forall u \in S \;.\; tu \in T\}$$

Keeping in mind that we wish to express each reducibility candidate R as $\mathcal{E} \to \mathtt{SN}$ for some $\mathcal{E}$, one could try to define the collection of reducibility candidates as the smallest set that contains $\mathtt{SN}$ and is closed under functional construction and arbitrary intersection. But then $\{\mathtt{nrec}\ \Omega\ \Omega\ \Box\} \to \mathtt{SN} = \emptyset$ is a valid candidate too. To avoid this we should be a bit more careful.

Definition 4.4.14. *Let $\mathcal{R}$ denote the collection of , this is the smallest collection of sets of terms satisfying the following conditions.*

(sn) $\mathtt{SN} \in \mathcal{R}$

($\bigcap$) *If $\mathbf{R} \subseteq \mathcal{R}$, then $\bigcap \mathbf{R} \in \mathcal{R}$.*

(app) *If $S, T \in \mathcal{R}$, then $S \to T \in \mathcal{R}$.*

(suc) *If $T \in \mathcal{R}$, then $\{\mathtt{S}\Box\} \to T \in \mathcal{R}$.*

(nrec) *If $S, T \in \mathcal{R}$, then $\{\mathtt{nrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T \in \mathcal{R}$.*

Lemma 4.4.15. *For each context E and variable x such that $E[x] \to t$ we have $t \equiv E'[x]$ and $E \to E'$.*

Proof. Simultaneous with $E[x] \not\equiv \lambda y.s$, $E[x] \not\equiv \mu\beta.d$ and $E[x] \not\equiv \underline{m}$ by induction on the structure of E. □

Lemma 4.4.16. *For each $R \in \mathcal{R}$ we have the following.*

1. *$R \subseteq \mathtt{SN}$*
2. *$E[x] \in R$ for each x and $E \in \mathtt{SN}^\Box$*

Proof. Simultaneously by induction on the generation of R.

(sn) Suppose that $R = \mathtt{SN}$. Now we certainly have $R \subseteq \mathtt{SN}$ and moreover $E[x] \in \mathtt{SN}$ by Lemma 4.4.15.

($\bigcap$) Suppose that $R = \bigcap \mathbf{R}$. Now we have $T \subseteq \mathtt{SN}$ for each $T \in \mathbf{R}$ by the induction hypothesis, hence $\bigcap \mathbf{R} \subseteq \mathtt{SN}$.

(app) Suppose that $R = S \to T$. For the first property, moreover suppose that $t \in R$, which means $tu \in T$ for each $u \in S$. The induction hypothesis guarantees that $x \in S$ and thus $tx \in T$. Therefore we have $tx \in \mathtt{SN}$ by the induction hypothesis, so certainly $t \in \mathtt{SN}$.

For the second property we have to show that $E[x] \in R$. So suppose that we are given a $u \in S$, now by the induction hypothesis we obtain that $u \in \mathtt{SN}$, hence the induction hypothesis of the second property guarantees that $E(\Box u)[x] \equiv E[x]u \in T$ and thus $E[x] \in R$.

(suc) Suppose that $R = \{\mathtt{S}\Box\} \to T$. For the first property, moreover suppose that $t \in R$, which means that $\mathtt{S}t \in T$. Therefore $\mathtt{S}t \in \mathtt{SN}$ by the induction hypothesis, so certainly $t \in \mathtt{SN}$.

For the second property we have to show that $E[x] \in R$. By the induction hypothesis we have $\mathtt{S}E[x] \in T$ and thus $E[x] \in R$.

(nrec) Suppose that $R = \{\mathtt{nrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T$. For the first property, moreover suppose that $t \in R$, which means that $\mathtt{nrec}\ r\ s\ t \in T$ for each $r \in T$ and $s \in S \to T \to T$. By the induction hypothesis we have $x \in T$ and $y \in S \to T \to T$ (by a similar argument as (2)), hence $\mathtt{nrec}\ x\ y\ t \in T \subseteq \mathtt{SN}$ and therefore certainly $t \in \mathtt{SN}$.

For the second property we have to show that $E[x] \in R$. Now we have $\mathtt{nrec}\ r\ s\ E[x] \in T$ for every $r \in T \subseteq \mathtt{SN}$ and $s \in S \to T \to T \subseteq \mathtt{SN}$ by the induction hypothesis, hence $E[x] \in R$. □

As we have remarked before, we wish to express each reducibility candidate R as $\mathcal{E} \to \mathtt{SN}$ for some set of contexts $\mathcal{E}$. Now we will make that idea precise.

Definition 4.4.17. *Given an $R \in \mathcal{R}$, a set of contexts $R^\perp$ is inductively defined on the generation of R as follows.*

$$\begin{aligned}
\mathtt{SN}^\perp &:= \{\Box\} \\
(\textstyle\bigcap \mathbf{R})^\perp &:= \bigcup \{T^\perp \mid T \in \mathbf{R}\} \\
(S \to T)^\perp &:= \{\Box\} \cup \{E(\Box u) \mid u \in S, E \in T^\perp\} \\
(\{\mathtt{S}\Box\} \to T)^\perp &:= \{\Box\} \cup \{E(\mathtt{S}\Box) \mid E \in T^\perp\} \\
(\{\mathtt{nrec}\ r\ s\ \Box\} \to T)^\perp &:= \{\Box\} \cup \{E(\mathtt{nrec}\ r\ s\ \Box) \mid r \in T, s \in S \to T \to T, E \in T^\perp\}
\end{aligned}$$

Fact 4.4.18. *For each $R \in \mathcal{R}$ we have $\Box \in R^\perp$.*

Lemma 4.4.19. *For each $R \in \mathcal{R}$ we have $R = R^\perp \to \mathtt{SN}$*

Proof. By induction on the generation of R.

(sn) Suppose that $R = \mathtt{SN}$. We have $R = \{\Box\} \to \mathtt{SN}$, so we are done.

($\bigcap$) Suppose that $R = \bigcap \mathbf{R}$. Now we have $T = T^\perp \to \mathtt{SN}$ for each $T \in \mathbf{R}$ by the induction hypothesis and therefore the following.

$$\begin{aligned} R &= \bigcap \{T^\perp \to \mathtt{SN} \mid T \in \mathbf{R}\} \\ &= \bigcap \{t \mid \forall T \in \mathbf{R}, E \in T^\perp \,.\, E[t] \in \mathtt{SN}\} \\ &= \bigcup \{T^\perp \mid T \in \mathbf{R}\} \to \mathtt{SN} \end{aligned}$$

(app) Suppose that $R = S \to T$. Now we have $T = T^\perp \to \mathtt{SN}$ by the induction hypothesis and therefore the following.

$$\begin{aligned} R &= \{\Box u \mid u \in S\} \to T^\perp \to \mathtt{SN} \\ &= \{t \mid \forall u \in S \,.\, tu \in T^\perp \to \mathtt{SN}\} \\ &= \{t \mid \forall u \in S, E \in T^\perp \,.\, E[tu] \in \mathtt{SN}\} \\ &= \{t \mid \forall F \in \{E(\Box u) \mid u \in S, E \in T^\perp\} \,.\, F[t] \in \mathtt{SN}\} \qquad (\star) \\ &= \{t \mid t \in \mathtt{SN} \wedge \forall F \in \{E(\Box u) \mid u \in S, E \in T^\perp\} \,.\, F[t] \in \mathtt{SN}\} \\ &= \{t \mid \forall F \in \{\Box\} \cup \{E(\Box u) \mid u \in S, E \in T^\perp\} \,.\, F[t] \in \mathtt{SN}\} \\ &= \{\Box\} \cup \{E(\Box u) \mid u \in S, E \in T^\perp\} \to \mathtt{SN} \end{aligned}$$

(suc) Suppose that $R = \{\mathtt{S}\Box\} \to T$. Now we have $T = T^\perp \to \mathtt{SN}$ by the induction hypothesis and therefore the following.

$$\begin{aligned} R &= \{\mathtt{S}\Box\} \to T^\perp \to \mathtt{SN} \\ &= \{t \mid \forall E \in T^\perp \,.\, E[\mathtt{S}t] \in \mathtt{SN}\} \qquad (\star) \\ &= \{t \mid t \in \mathtt{SN} \wedge \forall E \in T^\perp \,.\, E[\mathtt{S}t] \in \mathtt{SN}\} \\ &= \{\Box\} \cup \{E(\mathtt{S}\Box) \mid E \in T^\perp\} \to \mathtt{SN} \end{aligned}$$

(nrec) Suppose that $R = \{\mathtt{nrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T$. Now we have $T = T^\perp \to \mathtt{SN}$ by the induction hypothesis and therefore the following.

$$\begin{aligned} R &= \{\mathtt{nrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T^\perp \to \mathtt{SN} \\ &= \{t \mid \forall E \in T^\perp, r \in T, s \in S \to T \to T \,.\, E[\mathtt{nrec}\ r\ s\ t] \in \mathtt{SN}\} \qquad (\star) \\ &= \{t \mid t \in \mathtt{SN} \wedge \forall E \in T^\perp, r \in T, s \in S \to T \to T \,.\, E[\mathtt{nrec}\ r\ s\ t] \in \mathtt{SN}\} \\ &= \{\Box\} \cup \{E(\mathtt{nrec}\ r\ s\ \Box) \mid r \in T, s \in S \to T \to T, E \in T^\perp\} \to \mathtt{SN} \end{aligned}$$

The steps ($\star$) are allowed because for each $T \in \mathcal{R}$ we have $T \neq \emptyset$ by Lemma 4.4.16, $T^\perp \neq \emptyset$ by Fact 4.4.18 and because $F[t] \in \mathtt{SN}$ implies $t \in \mathtt{SN}$. □

Lemma 4.4.20. *For each $R \in \mathcal{R}$ and term t we have $t \in R$ iff $E[t] \in \mathtt{SN}$ for all contexts $E \in R^\perp$.*

Proof. Assume that $t \in R$ and $E \in R^\perp$. Now $R = R^\perp \to \mathtt{SN}$ by Lemma 4.4.19 and therefore $E[t] \in \mathtt{SN}$ by Definition 4.4.13.

For the reverse implication assume that $E[t] \in \mathtt{SN}$ for each $E \in R^\perp$. Now $t \in R^\perp \to \mathtt{SN}$ by Definition 4.4.13 and therefore $t \in R$ by Lemma 4.4.19. □

Now, before we can prove strong normalization of $\to_A$, it remains to give an interpretation $[\![\rho]\!] \in \mathcal{R}$ for each type ρ. Intuitively one would try to adapt the introduction for $\lambda\to$, which we have given in the introduction of this section.

$$[\![\mathtt{N}]\!] := \mathtt{SN}$$
$$[\![\rho \to \delta]\!] := [\![\delta]\!] \to [\![\sigma]\!]$$

Unfortunately, the interpretation of N does not contain enough structure to prove the following properties.

1. If $t \in \mathtt{SN}$, then $\mathtt{S}t \in \mathtt{SN}$.
2. If $t \in \mathtt{SN}$, $r \in S$ and $s \in \mathtt{SN} \to S \to S$, then $\mathtt{nrec}\ r\ s\ t \in S$.

Here, the term t could reduce to a term of the shape $\mu\alpha.c$ and is thereby able to consume the surrounding S or nrec. To define an interpretation of N that contains more structure we introduce the following definition first.

Definition 4.4.21. *Let $\mathcal{N}$ denote the smallest collection of terms satisfying the following conditions.*

(sn) $\mathtt{SN} \in \mathcal{N}$

(suc) *If $S \in \mathcal{N}$, then $\{\mathtt{S}\Box\} \to S \in \mathcal{N}$.*

(nrec) *If $S \in \mathcal{N}$ and $T \in \mathcal{R}$, then $\{\mathtt{nrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T \in \mathcal{N}$.*

Definition 4.4.22. *The* interpretation $[\![\rho]\!]$ *of a type ρ is defined as follows.*

$$[\![\mathtt{N}]\!] := \bigcap \mathcal{N}$$
$$[\![\delta \to \sigma]\!] := [\![\delta]\!] \to [\![\sigma]\!]$$

Fact 4.4.23. *For each type ρ we have $[\![\rho]\!] \in \mathcal{R}$.*

Lemma 4.4.24. *For each $n \in \mathbb{N}$ we have $\underline{n} \in [\![\mathtt{N}]\!]$.*

Proof. In order to prove this result we have to show that $\underline{n} \in R$ for all $R \in \mathcal{N}$ and $n \in \mathbb{N}$. We proceed by induction on the generation of R.

(var) Suppose that $R = \mathtt{SN}$. Now we have to show that $\underline{n} \in \mathtt{SN}$ for all $n \in \mathbb{N}$. However, $\underline{n}$ is in normal form, so we have certainly $\underline{n} \in \mathtt{SN}$.

(suc) Suppose that $R = \{\mathtt{S}\Box\} \to S$. Now we have $\underline{n} \in S$ for all $n \in \mathbb{N}$ by the induction hypothesis and have to show that $\mathtt{S}\underline{n} \in S$ for all $n \in \mathbb{N}$. However, $\mathtt{S}\underline{n} \equiv \underline{n+1}$, so the required result follows immediately from the induction hypothesis.

(nrec) Suppose that $R = \{\mathtt{nrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T$. Now we have $\underline{n} \in S$ for all $n \in \mathbb{N}$ by the induction hypothesis and have to show that $\mathtt{nrec}\ r\ s\ \underline{n} \in T$ for all $S \in \mathcal{N}$, $T \in \mathcal{R}$, $r \in T$, $s \in S \to T \to T$ and $n \in \mathbb{N}$. We proceed by induction on n.

(a) Suppose that $n = 0$. We have $E[r] \in \mathtt{SN}$ for all $E \in T^\perp$ by Lemma 4.4.20 and $s \in \mathtt{SN}$ by Lemma 4.4.16. Hence $E[\mathtt{nrec}\ r\ s\ 0] \in \mathtt{SN}$ by Lemma 4.4.12 and therefore $\mathtt{nrec}\ r\ s\ 0 \in T$ by Lemma 4.4.20.

(b) Suppose that $n > 0$. Now we have $\mathtt{nrec}\ r\ s\ \underline{n-1} \in T$ by the induction hypothesis. Furthermore, because $s \in S \to T \to T$ and $\underline{n-1} \in S$, we have $s\ \underline{n-1}\ (\mathtt{nrec}\ r\ s\ \underline{n-1}) \in T$ and therefore $E[s\ \underline{n-1}\ (\mathtt{nrec}\ r\ s\ \underline{n-1})] \in \mathtt{SN}$ for all $E \in T^\perp$ by Lemma 4.4.20. Hence $E[\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n-1})] \in \mathtt{SN}$ by Lemma 4.4.12, so $\mathtt{nrec}\ r\ s\ \underline{n} \in T$ by Lemma 4.4.20. □

Lemma 4.4.25. *If $t \in [\![\mathtt{N}]\!]$, then $\mathtt{S}t \in [\![\mathtt{N}]\!]$.*

Proof. Assume that $t \in [\![\mathtt{N}]\!]$, that means, $t \in R$ for all $R \in \mathcal{N}$. Now we have to prove that $\mathtt{S}t \in R$ for all $R \in \mathcal{N}$. But for all $R \in \mathcal{N}$ we have $\{\mathtt{S}\Box\} \to R \in \mathcal{N}$, hence $t \in \{\mathtt{S}\Box\} \to R$ by assumption and therefore $\mathtt{S}t \in R$. □

Lemma 4.4.26. *If $r \in [\![\rho]\!]$, $s \in [\![\mathbb{N} \to \rho \to \rho]\!]$ and $t \in [\![\mathbb{N}]\!]$, then $\mathtt{nrec}\ r\ s\ t \in [\![\rho]\!]$.*

Proof. We have $[\![\mathtt{N}]\!] \in \mathcal{N}$ by Definition 4.4.22, so if $t \in [\![\mathtt{N}]\!]$, then $\mathtt{nrec}\ r\ s\ t \in T$ for all $T \in \mathcal{R}$, $r \in T$ and $s \in [\![\mathtt{N}]\!] \to T \to T$ by Definition 4.4.21. Also $[\![\rho]\!] \in \mathcal{R}$ by Fact 4.4.23 and $[\![\mathtt{N} \to \rho \to \rho]\!] = [\![\mathtt{N}]\!] \to [\![\rho]\!] \to [\![\rho]\!]$, hence $\mathtt{nrec}\ r\ s\ t \in [\![\rho]\!]$. □

Lemma 4.4.27. *Let $x_1 : \rho_1, \ldots, x_n : \rho_n; \alpha_1 : \delta_1, \ldots, \alpha_m : \delta_m \vdash t : \sigma$ such that $r_i \in [\![\rho_i]\!]$ for all $1 \le i \le n$ and $E_j \in [\![\delta_j]\!]^\perp$ for all $1 \le j \le m$, then:*

$$t[x_1 := r_1, \ldots, x_1 := r_1, \alpha_1 := \alpha_1\ E_1, \ldots, \alpha_m := \alpha_m\ E_m] \in [\![\sigma]\!]$$

Proof. Abbreviate $\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n$, $\Delta = \alpha_1 : \delta_1, \ldots, \alpha_m : \delta_m$ and $t' \equiv t[x_1 := r_1, \ldots, x_1 := r_1, \alpha_1 := \alpha_1\ E_1, \ldots, \alpha_m := \alpha_m\ E_m]$. Now by mutual induction we prove that $\Gamma; \Delta \vdash t : \sigma$ implies $t \in [\![\sigma]\!]$ and that $\Gamma; \Delta \vdash c : \perp\!\!\!\perp$ implies $c' \in \mathtt{SN}^\perp$.

(var) Let $\Gamma; \Delta \vdash x : \sigma$ such that $x : \sigma \in \Gamma$. Now we have $x' \in [\![\sigma]\!]$ by assumption.

(λ) Let $\Gamma; \Delta \vdash \lambda x : \rho.t : \rho \to \delta$ with $\Gamma, x : \rho; \Delta \vdash t : \delta$. Moreover let $u \in [\![\rho]\!]$ and $E \in [\![\delta]\!]^\perp$. Now we have $t'[x := u] \in [\![\delta]\!]$ by the induction hypothesis and so $E[t'[x := u]] \in \mathtt{SN}$ by Lemma 4.4.20. Therefore $E[(\lambda x.t')u] \in \mathtt{SN}$ by Lemma 4.4.9 and hence $(\lambda x.t')u \in [\![\delta]\!]$ by Lemma 4.4.20, so $\lambda x.t' \in [\![\rho \to \delta]\!]$ by Definition 4.4.13.

(app) Let $\Gamma; \Delta \vdash ts : \delta$ with $\Gamma; \Delta \vdash t : \rho \to \delta$ and $\Gamma; \Delta \vdash s : \rho$. Now we have $t' \in [\![\rho \to \delta]\!] = [\![\rho]\!] \to [\![\delta]\!]$ and $s' \in [\![\rho]\!]$ by the induction hypothesis, hence $t's' \in [\![\delta]\!]$ by Definition 4.4.13.

(zero) Let $\Gamma; \Delta \vdash 0 : \mathtt{N}$. Now we have $0 \in [\![\mathtt{N}]\!]$ by Lemma 4.4.24.

(suc) Let $\Gamma; \Delta \vdash \mathtt{S}t : \mathtt{N}$ with $\Gamma; \Delta \vdash t : \mathtt{N}$. Now we have $t' \in [\![\mathtt{N}]\!]$ by the induction hypothesis and therefore $\mathtt{S}t' \in [\![\mathtt{N}]\!]$ by Lemma 4.4.25.

(nrec) Let $\Gamma; \Delta \vdash \mathtt{nrec}\ r\ s\ t : \rho$ with $\Gamma; \Delta \vdash r : \rho$, $\Gamma; \Delta \vdash s : \mathtt{N} \to \rho \to \rho$ and $\Gamma; \Delta \vdash t : \mathtt{N}$. Now we have $r' \in [\![\rho]\!]$, $s' \in [\![\mathtt{N} \to \rho \to \rho]\!]$ and $t' \in [\![\mathtt{N}]\!]$ by the induction hypothesis. Therefore $\mathtt{nrec}\ r'\ s'\ t' \in [\![\rho]\!]$ by Lemma 4.4.26.

(act) Let $\Gamma; \Delta \vdash \mu\alpha : \rho.c : \rho$ with $\Gamma; \Delta.\alpha : \rho \vdash c : \perp\!\!\!\perp$. Moreover let $E \in [\![\rho]\!]^\perp$. Now we have $c'[\alpha := \alpha E] \in \mathtt{SN}^\perp$ by the induction hypothesis. Hence $\mu\alpha.c'[\alpha := \alpha E] \in \mathtt{SN}$ and therefore $E[\mu\alpha.c'] \in \mathtt{SN}$ by Corollary 4.4.11, so $\mu\alpha.c' \in [\![\rho]\!]$ by Lemma 4.4.20.

(pas) Let $\Gamma;\Delta \vdash [\alpha]t : \perp\!\!\!\perp$ with $\alpha : \delta \in \Delta$ and $\Gamma;\Delta \vdash t : \delta$. Now we have $t' \in [\![\delta]\!]$ by the induction hypothesis and a context $E \in [\![\delta]\!]^\perp$ by assumption. Therefore $E[t'] \in \mathtt{SN}$ by Lemma 4.4.20 and so $[\alpha]E[t'] \in \mathtt{SN}^\perp$ because $([\alpha]t)' = [\alpha]E[t']$. □

Corollary 4.4.28. *If $\Gamma;\Delta \vdash t : \rho$, then $t \in \mathtt{SN}_A$.*

Proof. Suppose that $\Gamma;\Delta \vdash t : \rho$. Now we have $x \in [\![\rho]\!]$ for each $x : \rho \in \Gamma$ by Lemma 4.4.16 and $\Box \in [\![\delta]\!]^\perp$ for each $\alpha : \delta \in \Delta$ by Fact 4.4.18. Therefore $t \in [\![\rho]\!]$ by Lemma 4.4.27 and hence $t \in \mathtt{SN}_A$ by Fact 4.4.23 and Lemma 4.4.16. □

4.4.2 Strong normalization of (A) and (B)

Lemma 4.4.29. *For each term t it holds that $t \in \mathtt{SN}_B$.*

Proof. By performing a $\to_{\mu\eta}$ or $\to_{\mu i}$-reduction step on t, the term t reduces strictly in its size and therefore $\to_B$-reduction is strongly normalising. □

Lemma 4.4.30. *A single $\to_A$-reduction step can be advanced. That means, if $t_1 \to_B t_2 \to_A t_3$, then there is a t_4 such that the following diagram commutes.*

$$\begin{array}{ccc} t_1 & \xrightarrow{B} & t_2 \\ {\scriptstyle A}\,\vdots & & \downarrow{\scriptstyle A} \\ t_4 & \xrightarrow[AB]{} & t_3 \end{array}$$

Proof. We prove this lemma by distinguishing cases on $t_1 \to_B t_2$ and $t_2 \to_A t_3$, we treat some interesting cases.

1. Let $(\lambda x.t)r \to_B (\lambda x.t)r' \to_A \lambda x.t[x := r']$ with $r \to_B r'$. Now by Lemma 4.3.17 we have $t[x := r] \twoheadrightarrow_{AB} t[x := r']$, hence the following diagram commutes.
$$\begin{array}{ccc} (\lambda x.t)r & \xrightarrow{B} & (\lambda x.t)r' \\ {\scriptstyle A}\downarrow & & \downarrow{\scriptstyle A} \\ t[x := r] & \xrightarrow[AB]{} & t[x := r'] \end{array}$$

2. Let $E^s[\mu\alpha.c] \to_B F^s[\mu\alpha.c] \to_A \mu\alpha.c[\alpha := \alpha F^s]$ with $E^s \to_B F^s$. Now by Lemma 4.3.21 we have $c[\alpha := \alpha E^s] \twoheadrightarrow_{AB} c[\alpha := \alpha F^s]$, hence the following diagram commutes.
$$\begin{array}{ccc} E^s[\mu\alpha.c] & \xrightarrow{B} & F^s[\mu\alpha.c] \\ {\scriptstyle A}\downarrow & & \downarrow{\scriptstyle A} \\ \mu\alpha.c[\alpha := \alpha E^s] & \xrightarrow[AB]{} & \mu\alpha.c[\alpha := \alpha F^s] \end{array}$$

3. Let $(\mu\alpha.[\alpha]\lambda x.t)r \to_B (\lambda x.t)r \to_A t[x := r]$. Now the following diagram commutes.

$$\begin{array}{ccc} (\mu\alpha.[\alpha]\lambda x.t)r & \xrightarrow{\quad B \quad} & (\lambda x.t)r \\ \Big\downarrow{\scriptstyle A} & & \Big\downarrow{\scriptstyle A} \\ \mu\alpha.[\alpha](\lambda x.t)r & \xrightarrow[A]{} \mu\alpha.[\alpha]t[x := r] & \xrightarrow[B]{} t[x := r] \end{array}$$

4. Let $\mathtt{nrec}\ r\ s\ (\mu\alpha.[\alpha]\mathtt{S}\underline{n}) \to_B \mathtt{nrec}\ r\ s\ \mathtt{S}\underline{n} \to_A s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n})$. Now the following diagram commutes.

$$\begin{array}{ccc} \mathtt{nrec}\ r\ s\ (\mu\alpha.[\alpha]\mathtt{S}\underline{n}) & \xrightarrow{\quad B \quad} & \mathtt{nrec}\ r\ s\ \mathtt{S}\underline{n} \\ \Big\downarrow{\scriptstyle A} & & \Big\downarrow{\scriptstyle A} \\ \mu\alpha.[\alpha]\mathtt{nrec}\ r\ s\ \mathtt{S}\underline{n} & \xrightarrow[A]{} \mu\alpha.[\alpha](s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n})) & \xrightarrow[B]{} s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n}) \end{array}$$

5. Let $E^s[\mu\alpha.[\alpha]\mu\beta.c] \to_B E^s[\mu\alpha.c[\beta := \alpha\ \Box]] \to_A \mu\alpha.c[\beta := \alpha\ \Box][\alpha := \alpha E^s]$. Now the following diagram commutes by Lemma 4.2.12.

$$\begin{array}{ccc} E^s[\mu\alpha.[\alpha]\mu\beta.c] & \xrightarrow{\quad B \quad} & E^s[\mu\alpha.c[\beta := \alpha\ \Box]] \\ \Big\downarrow{\scriptstyle A} & & \Big\downarrow{\scriptstyle A} \\ \mu\alpha.[\alpha]E^s[\mu\beta.c[\alpha := \alpha E^s]] & & \mu\alpha.c[\beta := \alpha\ \Box][\alpha := \alpha E^s] \\ & {\scriptstyle A}\searrow \quad \swarrow{\scriptstyle B} & \\ & \mu\alpha.[\alpha]\mu\beta.c[\alpha := \alpha E^s][\beta := \beta E^s] & \end{array}$$

6. Let $E^s[\mu\alpha.[\gamma]\mu\beta.c] \to_B E^s[\mu\alpha.c[\beta := \gamma\ \Box]] \to_A \mu\alpha.c[\beta := \gamma\ \Box][\alpha := \alpha E^s]$ such that $\alpha \neq \gamma$. Now the following diagram commutes by Lemma 4.2.12.

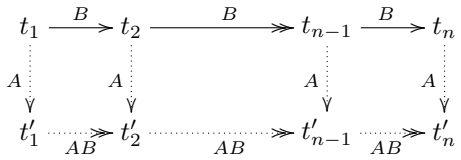
$$\begin{array}{ccc} E^s[\mu\alpha.[\gamma]\mu\beta.c] & \xrightarrow{\quad B \quad} & E^s[\mu\alpha.c[\beta := \gamma\ \Box]] \\ \Big\downarrow{\scriptstyle A} & & \Big\downarrow{\scriptstyle A} \\ \mu\alpha.[\gamma]\mu\beta.c[\alpha := \alpha E^s] & \xrightarrow[B]{} & \mu\alpha.c[\beta := \gamma\ \Box][\alpha := \alpha E^s] \end{array}$$

□

Corollary 4.4.31. *An $\to_A$-reduction step after multiple $\to_B$-reduction steps can be advanced. That means, if $t_1 \twoheadrightarrow_B t_2 \to_A t_3$, then there is a t_4 such that the following diagram commutes.*

$$\begin{array}{ccc} t_1 & \xrightarrow{B} & t_2 \\ {\scriptstyle A}\vdots & & \Big\downarrow{\scriptstyle A} \\ t_4 & \overset{AB}{\cdots\!\!\twoheadrightarrow} & t_3 \end{array}$$

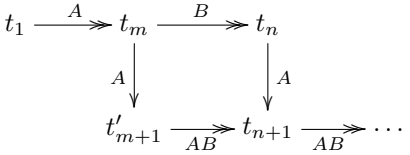
Proof. By iteration of Lemma 4.4.30 we have the following.



□

Lemma 4.4.32. *For each infinite reduction sequence $t_1 \twoheadrightarrow_A t_m \to \ldots$ that begins with m consecutive $\to_A$-reduction steps there exists an infinite reduction sequence $t_1 \twoheadrightarrow_A t_m \to_A t'_{m+1} \twoheadrightarrow \ldots$ that begins with least $m+1$ consecutive $\to_A$-reduction steps.*

Proof. Assume that we have an infinite reduction sequence $t_1 \twoheadrightarrow_A t_m \to \ldots$ If $t_m \to_A t_{m+1}$ we are immediately done, so let us assume that $t_m \to_B t_{m+1}$. Lemma 4.4.29 guarantees that an infinite sequence of $\to_B$-reduction steps does not exist, so there should be an $n > m$ such that $t_m \twoheadrightarrow_B t_n \to_A t_{n+1} \twoheadrightarrow \ldots$ By Corollary 4.4.31 we obtain a term t'_{m+1} such that the following diagram commutes.



This reduction sequence is also infinite and starts with at least $m+1$ consecutive $\to_A$-reduction steps, so we are done. □

Corollary 4.4.33. *For each infinite $\to_{AB}$-reduction sequence starting at t_1 there exists an infinite $\to_A$-reduction sequence starting at t_1.*

Proof. By iteration of Lemma 4.4.32. □

Theorem 4.4.34. *If $\Gamma; \Delta \vdash t : \rho$, then $t \in \mathtt{SN}$.*

Proof. Assume the contrary, then there should be an infinite $\to_{AB}$-reduction sequence. But then by Corollary 4.4.33 we can obtain an infinite $\to_A$ reduction sequence and therefore $t \notin \mathtt{SN}_A$, which contradicts Corollary 4.4.28. □

4.5 CPS-translation of $\lambda^{\mathbf{T}}_{\mu}$ into $\lambda^{\mathbf{T}}$

In this section we will present a CPS-translation from $\lambda^{\mathbf{T}}_{\mu}$ into $\lambda^{\mathbf{T}}$. We will use this CPS-translation to prove the main result of this section: the functions that are definable in $\lambda^{\mathbf{T}}_{\mu}$ are exactly the functions that are provably recursive in first-order arithmetic.

Definition 4.5.1. *Given a type τ, then let $\neg\rho$ denote $\rho \to \tau$. Now given a type ρ, then the* negative translation *ρ° of ρ is mutually inductively defined with $\rho^\bullet$ as follows.*

$$\begin{aligned} \rho^\circ &:= \neg\neg\rho^\bullet \\ \mathtt{N}^\bullet &:= \mathtt{N} \\ (\rho \to \delta)^\bullet &:= \rho^\circ \to \delta^\circ \end{aligned}$$

Definition 4.5.2. *Given $\lambda^{\boldsymbol{T}}$-terms t and r, then* the CPS-application *$t \bullet r$ of t and r is defined as follows.*

$$t \bullet r := \lambda k.t(\lambda l.lrk)$$

Definition 4.5.3. *Given a $\lambda^{\boldsymbol{T}}$-term t, then* the negative *of t is defined as follows.*

$$\underline{t} := \lambda k.kt$$

Lemma 4.5.4. *Given $\lambda^{\boldsymbol{T}}$-terms t and r such that $\Gamma \vdash t : (\rho \to \delta)^\circ$ and $\Gamma \vdash r : \rho^\circ$, then $\Gamma \vdash t \bullet r : \delta^\circ$.*

Proof. Suppose that $\Gamma \vdash t : (\rho \to \delta)^\circ$ and $\Gamma \vdash r : \rho^\circ$. Now we have $\Gamma \vdash t \bullet r : \delta^\circ$ as shown below.

$$\dfrac{t : (\rho \to \delta)^\circ \qquad \dfrac{\dfrac{\dfrac{l : \rho^\circ \to \delta^\circ \qquad r : \rho^\circ}{lr : \delta^\circ} \qquad k : \neg\delta^\bullet}{lrk : \bot}}{\lambda l.lrk : \neg(\rho^\circ \to \delta^\circ)}}{\dfrac{t(\lambda l.lrk) : \bot}{\lambda k.t(\lambda l.lrk) : \delta^\circ}}$$

□

Definition 4.5.5. *Given a $\lambda_\mu^{\mathbf{T}}$-term t, then the* CPS-translation *t° of t into $\lambda^{\boldsymbol{T}}$ is inductively defined as follows.*

$$\begin{aligned} x^\circ &:= \lambda k.xk \\ (\lambda x.t)^\circ &:= \lambda k.k(\lambda x.t^\circ) \\ (tr)^\circ &:= t^\circ \bullet r^\circ \\ 0^\circ &:= \underline{0} \\ (\mathtt{S}t)^\circ &:= \lambda k.t^\circ(\lambda l.k(\mathtt{S}l)) \\ (\mathtt{nrec}_\rho\ r\ s\ t)^\circ &:= \lambda k.t^\circ(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k) \\ &\quad \textit{where } s' := \lambda xp.s^\circ \bullet \underline{x} \bullet p \\ (\mu\alpha.c)^\circ &:= \lambda k_\alpha.c^\circ \\ ([\alpha]t)^\circ &:= t^\circ k_\alpha \end{aligned}$$

Here k_α is a fresh λ-variable for each μ-variable α.

In the translation of $\mathtt{nrec}_\rho\ r\ s\ t$ we see that we are required to evaluate t first, simply because it is the only way to obtain a numeral from t.

Lemma 4.5.6. *Given a $\lambda^{\boldsymbol{T}}$-term t such that $\Gamma \vdash t : \mathtt{N}$, then $\Gamma \vdash \underline{t} : \mathtt{N}^\circ$.*

Proof. Suppose that $\Gamma \vdash t : \mathtt{N}$. Now we have $\Gamma \vdash \underline{t} : \mathtt{N}^\circ$ as shown below.

$$\dfrac{\dfrac{k : \neg\mathtt{N} \qquad t : \mathtt{N}}{kt : \bot}}{\lambda k.kt : \mathtt{N}^\circ}$$

□

Theorem 4.5.7. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda^{\boldsymbol{T}}$ preserves typing. That is:*

$$\Gamma; \Delta \vdash t : \rho \text{ in } \lambda_\mu^{\mathbf{T}} \qquad \Longrightarrow \qquad \Gamma^\circ, \Delta^\circ \vdash t^\circ : \rho^\circ \text{ in } \lambda^{\boldsymbol{T}}$$

where $\Gamma^\circ = \{x : \rho^\circ \mid x : \rho \in \Gamma\}$ *and* $\Delta^\circ = \{k_\alpha : \neg\rho^\bullet \mid \alpha : \rho \in \Delta\}$.

Proof. We prove that we have $\Gamma; \Delta \vdash t : \rho$ and $\Gamma; \Delta \vdash c : \perp\!\!\!\perp$ by mutual induction on the derivations $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \rho^\circ$ and $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \bot$, respectively. We treat some interesting cases.

(zero) Let $\Gamma; \Delta \vdash 0 : \mathtt{N}$. Now we have $\Gamma^\circ, \Delta^\circ \vdash 0^\circ : \mathtt{N}^\circ$ by Lemma 4.5.6.

(suc) Let $\Gamma; \Delta \vdash \mathtt{S}t : \mathtt{N}$ with $\Gamma; \Delta \vdash t : \mathtt{N}$. Now we have $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \mathtt{N}$ by the induction hypothesis. So $\Gamma^\circ, \Delta^\circ \vdash (\mathtt{S}t)^\circ : \mathtt{N}^\circ$ as shown below.

$$\dfrac{t^\circ : \mathtt{N}^\circ \qquad \dfrac{k : \neg\mathtt{N} \qquad \dfrac{l : \mathtt{N}}{\mathtt{S}l : \mathtt{N}}}{k(\mathtt{S}l) : \bot}}{\dfrac{t^\circ(\lambda l.k(\mathtt{S}l)) : \bot}{\lambda k.t^\circ(\lambda l.k(\mathtt{S}l)) : \mathtt{N}^\circ}}$$

(nrec) Let $\Gamma; \Delta \vdash \mathtt{nrec}_\rho\ r\ s\ t : \rho$ with $\Gamma; \Delta \vdash r : \rho$, $\Gamma; \Delta \vdash s : \mathtt{N} \to \rho \to \rho$ and $\Gamma; \Delta \vdash t : \mathtt{N}$. Now we have $\Gamma^\circ, \Delta^\circ \vdash r^\circ : \rho^\circ$, $\Gamma^\circ, \Delta^\circ \vdash s^\circ : (\mathtt{N} \to \rho \to \rho)^\circ$ and $\Gamma^\circ, \Delta^\circ \vdash t^\circ : \mathtt{N}^\circ$ by the induction hypothesis. Furthermore we have $s' : \mathtt{N} \to \rho^\circ \to \rho^\circ$ as shown below.

$$\dfrac{\dfrac{\dfrac{s^\circ : (\mathtt{N} \to \rho \to \rho)^\circ \qquad \dfrac{x : \mathtt{N}}{\underline{x} : \mathtt{N}^\circ}\,(c)}{s^\circ \bullet \underline{x} : (\rho \to \rho)^\circ}\,(b) \qquad p : \rho^\circ}{s^\circ \bullet \underline{x} \bullet p : \rho^\circ}\,(a)}{\lambda x p.s^\circ \bullet \underline{x} \bullet p : \mathtt{N} \to \rho^\circ \to \rho^\circ}$$

Here, step (a) follows from Lemma 4.5.6 and step (b) and (c) follows from Lemma 4.5.4. So $\Gamma^\circ, \Delta^\circ \vdash (\mathtt{nrec}_\rho\ r\ s\ t)^\circ : \rho^\circ$ as shown below.

$$\dfrac{t^\circ : \mathtt{N}^\circ \qquad \dfrac{\dfrac{\dfrac{r^\circ : \rho^\circ \qquad s' : \mathtt{N} \to \rho^\circ \to \rho^\circ \qquad l : \mathtt{N}}{\mathtt{nrec}\ r^\circ\ s'\ l : \rho^\circ} \qquad k : \neg\rho^\bullet}{\mathtt{nrec}\ r^\circ\ s'\ l\ k : \bot}}{\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k : \neg\mathtt{N}}}{\dfrac{t^\circ(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k) : \bot}{\lambda k.t^\circ(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k) : \rho^\circ}}$$

□

Lemma 4.5.8. *Given a natural number n, then $\underline{n}^\circ \twoheadrightarrow \underline{\underline{n}}$.*

Proof. By induction on n.

1. Suppose that $n = 0$. Now we have $\underline{0}^\circ \equiv \underline{\underline{0}}$ by definition.
2. Suppose that $n > 0$. Now we have $\underline{n}^\circ \twoheadrightarrow \underline{\underline{n}}$ by the induction hypothesis and therefore:

$$\begin{aligned}\underline{n+1}^\circ &\equiv \lambda k.\underline{n}^\circ(\lambda l.k(\mathtt{S}l)) \\ &\twoheadrightarrow \lambda k.(\lambda q.q\underline{n})(\lambda l.k(\mathtt{S}l)) \\ &\twoheadrightarrow \lambda k.k(\mathtt{S}\underline{n}) \\ &\equiv \underline{\underline{n+1}}\end{aligned}$$

□

Lemma 4.5.9. *Given a $\lambda_\mu^{\mathbf{T}}$-term t, then $\lambda k.t^\circ k \to t^\circ$.*

Proof. This result is proven similarly to Lemma 3.5.6. □

Lemma 4.5.10. *Given $\lambda_\mu^{\mathbf{T}}$-terms r and s, and moreover let $s' = \lambda xp.s^\circ \bullet \underline{x} \bullet p$, then $\lambda k.\mathtt{nrec}\ r^\circ\ s'\ \underline{n}\ k = \mathtt{nrec}\ r^\circ\ s'\ \underline{n}$.*

Proof. We distinguish the following cases.

1. Suppose that $n = 0$. Now we have:

$$\begin{aligned}\lambda k.\mathtt{nrec}\ r^\circ\ s'\ 0\ k &\twoheadrightarrow \lambda k.r^\circ k \\ &\twoheadrightarrow r^\circ \\ &= \mathtt{nrec}\ r^\circ\ s' \qquad \text{(a)}\end{aligned}$$

   Here, step (a) holds by Lemma 4.5.9.

2. Suppose that $n > 0$. Now we have:

$$\begin{aligned}\lambda k.\mathtt{nrec}\ r^\circ\ s'\ \underline{n}\ k &\twoheadrightarrow \lambda k.s'\ \underline{n-1}\ (\mathtt{nrec}\ r^\circ\ s'\ \underline{n-1})\ k \\ &\twoheadrightarrow \lambda k.(s^\circ \bullet \underline{\underline{n-1}} \bullet \mathtt{nrec}\ r^\circ\ s'\ \underline{n-1})\ k \\ &\equiv \lambda k.(\lambda k_2.(s^\circ \bullet \underline{\underline{n-1}})\ (\lambda l.l\ (\mathtt{nrec}\ r^\circ\ s'\ \underline{n-1})\ k_2))\ k \\ &\twoheadrightarrow \lambda k.(s^\circ \bullet \underline{\underline{n-1}})\ (\lambda l.l\ (\mathtt{nrec}\ r^\circ\ s'\ \underline{n-1})\ k) \\ &\equiv s^\circ \bullet \underline{\underline{n-1}} \bullet \mathtt{nrec}\ r^\circ\ s'\ \underline{n-1} \\ &= s'\ \underline{n-1}\ (\mathtt{nrec}\ r^\circ\ s'\ \underline{n-1}) \\ &= \mathtt{nrec}\ r^\circ\ s'\ \underline{n}\end{aligned}$$

□

Lemma 4.5.11. *Given $\lambda_\mu^{\mathbf{T}}$-terms t and r, then $t^\circ[x := r^\circ] \twoheadrightarrow (t[x := r])^\circ$.*

Proof. This result is proven similarly to Lemma 3.5.7. □

Lemma 4.5.12. *Given a $\lambda_\mu^{\mathbf{T}}$-term t, then:*

1. $(t[\alpha := \beta\ \Box])^\circ \equiv t^\circ[k_\alpha := k_\beta]$
2. $(t[\alpha := \beta\ (\mathtt{S}\Box)])^\circ \twoheadrightarrow t^\circ[k_\alpha := \lambda l.k_\beta(\mathtt{S}l)]$

3. $(t[\alpha := \beta\ (\Box s)])^\circ \twoheadrightarrow t^\circ[k_\alpha := \lambda l.ls^\circ k_\beta]$
4. $(t[\alpha := \beta\ (\mathtt{nrec}\ r\ s\ \Box)])^\circ \twoheadrightarrow t^\circ[k_\alpha := \lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k_\beta]$

Proof. This result is proven similarly to Lemma 3.5.8. □

Lemma 4.5.13. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda^{\boldsymbol{T}}$ preserves equality. That is, given $\lambda_\mu^{\mathbf{T}}$-terms t_1 and t_2 such that $t_1 = t_2$, then ${t_1}^\circ = {t_2}^\circ$.*

Proof. By induction on $t_1 \to t_2$. We treat some interesting cases.

1. Let $\mathtt{S}(\mu\alpha.c) \to \mu\alpha c[\alpha := \alpha\ (\mathtt{S}\Box)]$. Now:

$$\begin{aligned}(\mathtt{S}(\mu\alpha.c))^\circ &\equiv \lambda k.(\lambda k_\alpha.c^\circ)\lambda l.k(\mathtt{S}l)\\ &\twoheadrightarrow \lambda k.c^\circ[k_\alpha := \lambda l.k(\mathtt{S}l)]\\ &= \lambda k_\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)]^\circ \qquad \text{(a)}\\ &\equiv (\mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)])^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 4.5.12.

2. Let $\mathtt{nrec}\ r\ s\ 0 \to r$. Now:

$$\begin{aligned}(\mathtt{nrec}\ r\ s\ 0)^\circ &\equiv \lambda k.\underline{0}(\lambda n.\mathtt{nrec}\ r^\circ\ s'\ n\ k)\\ &\twoheadrightarrow \lambda k.\mathtt{nrec}\ r^\circ\ s'\ 0\ k\\ &\to \lambda k.r^\circ k \qquad \text{(a)}\\ &= r^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 4.5.9.

3. Let $\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n}) \to s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n})$. Now:

$$\begin{aligned}(\mathtt{nrec}\ r\ s\ (\mathtt{S}\underline{n}))^\circ &\equiv \lambda k.(\mathtt{S}\underline{n})^\circ(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k)\\ &\twoheadrightarrow \lambda k.\underline{\mathtt{S}\underline{n}}(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k) \qquad \text{(a)}\\ &\twoheadrightarrow \lambda k.\mathtt{nrec}\ r^\circ\ s'\ (\mathtt{S}\underline{n})\ k\\ &\to \lambda k.s'\ \underline{n}\ (\mathtt{nrec}\ r^\circ\ s'\ \underline{n})\ k\\ &\twoheadrightarrow \lambda k.(s^\circ \bullet \underline{\underline{n}} \bullet \mathtt{nrec}\ r^\circ\ s'\ \underline{n})\ k\\ &= \lambda k.(\lambda k_2.(s^\circ \bullet \underline{\underline{n}})\ (\lambda l.l\ (\mathtt{nrec}\ r^\circ\ s'\ \underline{n})\ k_2))\ k\\ &= \lambda k.(s^\circ \bullet \underline{\underline{n}})\ (\lambda l.l\ (\mathtt{nrec}\ r^\circ\ {}'s\ \underline{n})\ k)\\ &= s^\circ \bullet \underline{\underline{n}} \bullet \mathtt{nrec}\ r^\circ\ s'\ \underline{n}\\ &= s^\circ \bullet \underline{\underline{n}} \bullet \lambda k_2.\mathtt{nrec}\ r^\circ\ s'\ \underline{n}\ k_2 \qquad \text{(b)}\\ &= s^\circ \bullet \underline{\underline{n}} \bullet \lambda k_2.\underline{n}(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k_2)\\ &= s^\circ \bullet \underline{\underline{n}} \bullet \lambda k_2.\underline{n}^\circ(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k_2) \qquad \text{(c)}\\ &\equiv (s\ \underline{n}\ (\mathtt{nrec}\ r\ s\ \underline{n}))^\circ\end{aligned}$$

   Here, step (a) holds by Lemma 4.5.8, step (b) holds by Lemma 4.5.10 and step (c) holds by Lemma 4.5.8.

4. Let $\mathtt{nrec}\ r\ s\ \mu\alpha.c \to \mu\alpha c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)]$. Now:

$$\begin{aligned}(\mathtt{nrec}\ r\ s\ \mu\alpha.c)^\circ &\equiv \lambda k.(\lambda k_\alpha.c^\circ)(\lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k)\\ &\twoheadrightarrow \lambda k.c^\circ[k_\alpha := \lambda l.\mathtt{nrec}\ r^\circ\ s'\ l\ k]\\ &= \lambda k_\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)]^\circ \qquad \text{(a)}\\ &\equiv (\mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)])^\circ\end{aligned}$$

Here, step (a) holds by Lemma 4.5.12. □

Theorem 4.5.14. *Each function $f : \mathbb{N}^n \to \mathbb{N}$ that is representable in $\lambda_\mu^{\mathbf{T}}$ is representable in $\lambda^{\boldsymbol{T}}$. That is, let a term $t : \mathtt{N}^n \to \mathtt{N}$ represent the function f in $\lambda_\mu^{\mathbf{T}}$, then there exists a term $t' : \mathtt{N}^n \to \mathtt{N}$ that represents the function f in $\lambda^{\boldsymbol{T}}$.*

Proof. Suppose that $t : \mathtt{N}^n \to \mathtt{N}$ represents $f : \mathbb{N}^n \to \mathbb{N}$ in $\lambda_\mu^{\mathbf{T}}$. That means that $\underline{f(m_1, \ldots, m_n)} = t\ \underline{m_1} \ldots \underline{m_n}$. Now define a term t' as follows.

$$t' := \lambda x_1 : \mathtt{N} \ldots \lambda x_n : \mathtt{N}\ .\ (t^\circ \bullet \underline{x_1} \bullet \ldots \bullet \underline{x_n})\ (\lambda x : \mathtt{N}\ .\ x)$$

Now we have $t^\circ : (\mathtt{N}^n \to \mathtt{N})^\circ$ by Theorem 4.5.7, $x_i : \mathtt{N}^\circ$ by Lemma 4.5.6 and therefore $t^\circ \bullet \underline{x_1} \bullet \ldots \bullet \underline{x_n} : \mathtt{N}^\circ$ by Lemma 4.5.4. Hence by letting $\bot = \mathtt{N}$ we have $t' : \mathtt{N}$. Now it remains to prove that $\underline{f(m_1, \ldots, m_n)} = t'\ \underline{m_1} \ldots \underline{m_n}$.

$$\begin{aligned}t'\ \underline{m_1} \ldots \underline{m_n} &= (t^\circ \bullet \underline{\underline{m_1}} \bullet \ldots \bullet \underline{\underline{m_n}})\ \lambda x.x\\ &= (t^\circ \bullet \underline{m_1}^\circ \bullet \ldots \bullet \underline{m_n}^\circ)\ \lambda x.x \qquad \text{(a)}\\ &= (t\ \underline{m_1} \ldots \underline{m_n})^\circ\ \lambda x.x\\ &= (\underline{f(m_1, \ldots, m_n)})^\circ\ \lambda x.x \qquad \text{(b)}\\ &= \underline{\underline{f(m_1, \ldots, m_n)}}\ \lambda x.x \qquad \text{(c)}\\ &= \underline{f(m_1, \ldots, m_n)}\end{aligned}$$

Here, step (a) holds by Lemma 4.5.8, step (b) holds by Lemma 4.5.13 and step (c) holds by Lemma 4.5.8. □

Corollary 4.5.15. *The functions definable in $\lambda_\mu^{\mathbf{2}}$ are exactly those that are provably recursive in first-order arithmetic.*

Proof. This result follows immediately from Theorem 4.1.12 and 4.5.14. □

4.6 Embedding $\lambda_\mu^{\mathbf{T}}$ into $\lambda_\mu^{\mathbf{2}}$

It is well known that the equational theory of $\lambda^{\mathbf{T}}$ can be embedded into $\lambda^{\mathbf{2}}$ [SU06]. So, using the translation in the preceding section, $\lambda_\mu^{\mathbf{T}}$ can be embedded into $\lambda^{\mathbf{T}}$ and thereby also in $\lambda_\mu^{\mathbf{2}}$. However, simulation of the control operator μ by CPS is not quite satisfactory since $\lambda_\mu^{\mathbf{2}}$ contains that operator too.

In this section we present a direct translation of $\lambda_\mu^{\mathbf{T}}$ into $\lambda_\mu^{\mathbf{2}}$. This translation has two goals. Firstly, it preserves occurences of the control operator μ. Secondly, it show how the output operator Φ (Definition 3.4.7) can be adapted, which nicely contrasts the differences between $\lambda_\mu^{\mathbf{T}}$ and $\lambda_\mu^{\mathbf{2}}$.

Definition 4.6.1. *Given a $\lambda^{\mathbf{T}}$-type δ, then a $\lambda^{\mathbf{2}}$-type $\delta^\diamond$ is inductively defined as follows.*

$$\begin{aligned} \mathtt{N}^\diamond &:= \mathsf{N} \\ (\rho \to \delta)^\diamond &:= \rho^\diamond \to \delta^\diamond \end{aligned}$$

Definition 4.6.2. *Given a $\lambda_\mu^{\mathbf{T}}$-term t, then a $\lambda_\mu^{\mathbf{2}}$-term $t^\diamond$ is inductively defined as follows.*

$$\begin{aligned} x^\diamond &:= x \\ 0^\diamond &:= c_0 \\ (\mathsf{S}t)^\diamond &:= t^\diamond\ ((\mathsf{N} \to \mathsf{N}) \to \mathsf{N})\ \hat{\mathsf{S}}\ \hat{0}\ (\lambda l.\mathsf{S}l) \\ &\quad \textit{where } \hat{0} := \lambda k.kc_0 \textit{ and } \hat{\mathsf{S}} := \lambda kh.k\,(\lambda l.h\,(\mathsf{S}l)) \\ (\lambda x.t)^\diamond &:= \lambda x.t^\diamond \\ (tr)^\diamond &:= t^\diamond r^\diamond \\ (\mathtt{nrec}_\rho\ r\ s\ t)^\diamond &:= \mathsf{nrec}_\rho\ r^\diamond\ s^\diamond\ t^\diamond \\ (\mu\alpha.c)^\diamond &:= \mu\alpha.c^\diamond \\ ([\alpha]t)^\diamond &:= [\alpha]t^\diamond \end{aligned}$$

The key idea of this embedding is the translation of the successor. This translation ensures that we do not create closed terms of type N whose normal form is of another shape than c_n. If we would translate $\mathsf{S}t$ into $\mathsf{S}t^\diamond$ instead, this property fails. For example, the normal form of $\mu\alpha.[\alpha]\mathsf{S}\Theta[\alpha]0$ is 0, but the normal of $(\mu\alpha.[\alpha]\mathsf{S}\Theta[\alpha]0)^\diamond$ is not a numeral.

$$\begin{aligned} (\mu\alpha.[\alpha]\mathsf{S}\Theta[\alpha]0)^\diamond &\equiv \mu\alpha.[\alpha](\lambda n.\lambda\rho.\lambda fx.f(n\rho fx))\,\Theta[\alpha]c_0 \\ &\twoheadrightarrow \mu\alpha.[\alpha]\lambda\rho.\lambda fx.f((\Theta[\alpha]c_0)\rho fx) \\ &\twoheadrightarrow \mu\alpha.[\alpha]\lambda\rho.\lambda fx.f(\Theta[\alpha]c_0) \end{aligned}$$

In Section 3.4, we have seen how Parigot [Par93] uses the output operator Φ to extract a numeral from such a term. Here we adapt the output operator for the translation of the successor. However, instead of using the identity function as the top continuation, we use $\lambda l.\mathsf{S}l$. The following lemma states that the translation of numerals is correct.

Lemma 4.6.3. *Given a natural number n, then $\underline{n}^\diamond \twoheadrightarrow c_n$.*

Proof. By induction on n.

1. Suppose that $n = 0$. Now we have $\underline{0}^\diamond \equiv c_0$ by definition.
2. Suppose that $n > 0$. Now we have $\underline{n}^\diamond \twoheadrightarrow c_n$ by the induction hypothesis and therefore:

$$\begin{aligned} \underline{n+1}^\diamond &\equiv (\mathsf{S}\underline{n})^\diamond \\ &\equiv \underline{n}^\diamond\ ((\mathsf{N} \to \mathsf{N}) \to \mathsf{N})\ \hat{\mathsf{S}}\ \hat{0}\ (\lambda l.\mathsf{S}l) \\ &\twoheadrightarrow c_n\ ((\mathsf{N} \to \mathsf{N}) \to \mathsf{N})\ \hat{\mathsf{S}}\ \hat{0}\ (\lambda l.\mathsf{S}l) \\ &\twoheadrightarrow (\lambda k.kc_n)\ (\lambda l.\mathsf{S}l) \qquad \text{(a)} \\ &\twoheadrightarrow c_{n+1} \end{aligned}$$

Here, step (a) is proven by induction on n. □

Lemma 4.6.4. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda_\mu^{\mathbf{2}}$ preserves typing. That is:*

$$\Gamma; \Delta \vdash t : \rho \text{ in } \lambda_\mu^{\mathbf{T}} \qquad \Longrightarrow \qquad \Gamma^\diamond; \Delta^\diamond \vdash t^\diamond : \rho^\diamond \text{ in } \lambda_\mu^{\mathbf{2}}$$

where $\Gamma^\diamond = \{x : \rho^\diamond \mid x : \rho \in \Gamma\}$ and $\Delta^\diamond = \{\alpha : \rho^\diamond \mid \alpha : \rho \in \Delta\}$.

Proof. Straightforward by induction on the derivation $\Gamma; \Delta \vdash t : \rho$. □

Lemma 4.6.5. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda_\mu^{\mathbf{2}}$ preserves $\to_{\mu\beta}$, $\to_{\mu\mathtt{S}}$, $\to_{\mu R}$, $\to_{\mu\eta}$, $\to_{\mu i}$ and $\to_{\mu\mathtt{N}}$-reduction. That is, if $t \to_{\mu\mathtt{S}} t'$, $t \to_{\mu R} t'$, $t \to_{\mu\eta} t'$, $t \to_{\mu i} t'$ or $t \to_{\mu\mathtt{N}} t'$, then $t^\diamond \twoheadrightarrow^+ t'^\diamond$.*

Proof. We treat some non-trivial cases.

1. Let $\mathtt{S}\mu\alpha.c \to \mu\alpha.t[\alpha := \alpha\ (\mathtt{S}\Box)]$. Now:

$$\begin{aligned} t^\diamond &\equiv (\mu\alpha.c^\diamond)\ ((\mathsf{N} \to \mathsf{N}) \to \mathsf{N})\ \hat{\mathsf{S}}\ \hat{0}\ (\lambda l.\mathsf{S}l) \\ &\twoheadrightarrow \mu\alpha.c^\diamond[\alpha := \alpha\ (\Box\ ((\mathsf{N} \to \mathsf{N}) \to \mathsf{N})\ \hat{\mathsf{S}}\ \hat{0}\ (\lambda l.\mathsf{S}l))] \\ &\equiv (\mu\alpha.t[\alpha := \alpha\ (\mathtt{S}\Box)])^\diamond \end{aligned}$$

2. Let $\mathtt{nrec}_\rho\ r\ s\ \mu\alpha.c \to \mu\alpha.t[\alpha := \alpha\ (\mathtt{nrec}_\rho\ r\ s\ \Box)]$. Now:

$$\begin{aligned} t^\diamond &\equiv \pi_1(\mu\alpha.c^\diamond\ (\rho \times \mathsf{N})\ (\lambda h.\langle s^\diamond\ (\pi_1 h)\ (\pi_2 h), \mathtt{S}(\pi_2 h)\rangle)\ \langle r^\diamond, c_0\rangle) \\ &\twoheadrightarrow \pi_1(\mu\alpha.c^\diamond[\alpha := \alpha\ (\Box\ (\rho \times \mathsf{N})\ (\lambda h.\langle s^\diamond\ (\pi_1 h)\ (\pi_2 h), \mathtt{S}(\pi_2 h)\rangle)\ \langle r^\diamond, c_0\rangle)]) \\ &\twoheadrightarrow \mu\alpha.c^\diamond[\alpha := \alpha\ \pi_1(\Box\ (\rho \times \mathsf{N})\ (\lambda h.\langle s^\diamond\ (\pi_1 h)\ (\pi_2 h), \mathtt{S}(\pi_2 h)\rangle)\ \langle r^\diamond, c_0\rangle))] \\ &\equiv (\mu\alpha.t[\alpha := \alpha\ (\mathtt{nrec}_\rho\ r\ s\ \Box)])^\diamond \end{aligned}$$

□

Lemma 4.6.6. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda_\mu^{\mathbf{2}}$ preserves primitive recursion That is:*

$$\begin{aligned} \mathsf{nrec}\ r^\diamond\ s^\diamond\ 0 &\twoheadrightarrow^+ r^\diamond \\ \mathsf{nrec}\ r^\diamond\ s^\diamond\ (\mathsf{S}\underline{n}^\diamond) &\twoheadrightarrow^+ s^\diamond\ c_n\ (s^\diamond\ c_{n-1}\ \ldots\ (s^\diamond\ c_0\ r^\diamond)) \end{aligned}$$

Proof. It is straightforward to show that $\mathsf{nrec}\ r^\diamond\ s^\diamond\ 0 \twoheadrightarrow^+ r^\diamond$. Furthermore, by induction on n we have $\mathsf{nrec}\ r^\diamond\ s^\diamond\ (\mathsf{S}c_n) \twoheadrightarrow^+ s^\diamond\ c_n\ (s^\diamond\ c_{n-1}\ \ldots\ (s^\diamond\ c_0\ r^\diamond))$, so our goal holds by Lemma 4.6.3. □

Corollary 4.6.7. *The translation from $\lambda_\mu^{\mathbf{T}}$ into $\lambda_\mu^{\mathbf{2}}$ preserves equality. That is, given $\lambda_\mu^{\mathbf{T}}$-terms t_1 and t_2 such that $t_1 = t_2$, then ${t_1}^\diamond = {t_2}^\diamond$.*

Proof. This result follows immediately from Lemma 4.6.5 and 4.6.6. □

Theorem 4.6.8. *Each function $f : \mathbb{N}^n \to \mathbb{N}$ that is representable in $\lambda_\mu^{\mathbf{T}}$ is representable in $\lambda_\mu^{\mathbf{2}}$. That is, let a term $t : \mathsf{N}^n \to \mathsf{N}$ represent the function f in $\lambda_\mu^{\mathbf{T}}$, then there exists a term $t' : \mathsf{N}^n \to \mathsf{N}$ that represents the function f in $\lambda_\mu^{\mathbf{2}}$.*

Proof. Suppose that $t : \mathsf{N}^n \to \mathsf{N}$ represents $f : \mathbb{N}^n \to \mathbb{N}$ in $\lambda_\mu^{\mathbf{T}}$. That means that $\underline{f(m_1, \ldots, m_n)} = t\ \underline{m_1} \ldots \underline{m_n}$. Now let $t' := t^\diamond$. We have $t^\diamond : \mathsf{N}^n \to \mathsf{N}$ by Lemma 4.6.4, so it remains to prove that $c_{f(m_1,\ldots,m_n)} = t^\diamond\ c_{m_1} \ldots c_{m_n}$.

$$\begin{aligned} t^\diamond\ c_{m_1} \ldots c_{m_n} &= (t^\diamond \underline{m_1}^\diamond \ldots \underline{m_n}^\diamond) && \text{(a)} \\ &= (t\ \underline{m_1} \ldots \underline{m_n})^\circ \\ &= (\underline{f(m_1, \ldots, m_n)})^\circ && \text{(b)} \\ &= c_{f(m_1,\ldots,m_n)} \end{aligned}$$

Here, step (a) and (b) hold by Lemma 4.6.3. □

4.7 Correctness of programs

In this section we will consider a simple $\lambda_\mu^{\mathbf{T}}$-program and prove its correctness by equational reasoning. This program is similar to the list product program given in the introduction of Chapter 3. However, since $\lambda_\mu^{\mathbf{T}}$ does not have a list data type we consider a program that computes the product of a term $f : \mathtt{N} \to \mathtt{N}$ applied to each numeral in the sequence $(\underline{0}, \ldots, \underline{n})$ for some $n \in \mathbb{N}$. Again, our program will stop multiplying once a zero is encountered. Because we need multiplication, for which we need addition, we define these notions first.

Definition 4.7.1. *The term* `plus` *is defined as follows.*

$$\mathtt{plus} := \mathtt{nrec}\ (\lambda y.y)\ (\lambda xhy.\mathtt{S}(hy))$$

Notation 4.7.2. *As usual we use the infix notation $x + y$ for* `plus` $x\ y$.

Since `plus` is the well-known addition function we will not prove its correctness, that is $\underline{x} + \underline{y} = \underline{x + y}$. It is more interesting to look at its behavior when it is applied to a term of the shape Θc. In this case it is required that the term Θc propagates through the function.

Lemma 4.7.3. *Given a command c and term t, then:*

$$(\Theta c) + t = \Theta c$$

Proof. This follows directly from the rules $\to_{\mu\mathtt{N}}$ and $\to_{\mu R}$.

$$\begin{aligned} (\Theta c) + t &\equiv \mathtt{nrec}\ (\lambda y.y)\ (\lambda xhy.\mathtt{S}(hy))\ (\Theta c)\ t \\ &\to (\Theta c)t \\ &\to \Theta c \end{aligned}$$

□

Lemma 4.7.4. *Given a command c and a natural number n, then:*

$$\underline{n} + (\Theta c) = \Theta c$$

Proof. By induction on n.

1. Suppose that $n = 0$. Now we have the following.

$$\begin{aligned} \underline{0} + (\Theta c) &\equiv \mathtt{nrec}\ (\lambda y.y)\ (\lambda xhy.\mathtt{S}(hy))\ 0\ \Theta c \\ &\to (\lambda y.y)\Theta c \\ &\to \Theta c \end{aligned}$$

2. Suppose that $n > 0$. Now we have $\underline{n} + (\Theta c) = \Theta c$ by the induction hypothesis and hence:

$$\begin{aligned}\underline{n+1} + (\Theta c) &\equiv \mathtt{nrec}\ (\lambda y.y)\ (\lambda xhy.\mathtt{S}(hy))\ \mathtt{S}\underline{n}\ (\Theta c)\\ &\to (\lambda xhy.\mathtt{S}(hy))\ \underline{n}\ (\mathtt{nrec}\ (\lambda y.y)\ (\lambda xhy.\mathtt{S}(hy))\ \underline{n})\ (\Theta c)\\ &\twoheadrightarrow \mathtt{S}(\mathtt{nrec}\ (\lambda y.y)\ (\lambda xhy.\mathtt{S}(hy))\ \underline{n}\ (\Theta c))\\ &\equiv \mathtt{S}(\underline{n} + (\Theta c))\\ &= \mathtt{S}(\Theta c)\\ &\to \Theta c\end{aligned}$$

□

Now we use addition to define multiplication. Again we use the well-known multiplication function, so we certainly have $\underline{x} * \underline{y} = \underline{x * y}$.

Definition 4.7.5. *The term* `mult` *is defined as follows.*

$$\mathtt{mult} := \mathtt{nrec}\ (\lambda y.0)\ (\lambda xhy.y + (hy))$$

Notation 4.7.6. *As usual we use the infix notation $x * y$ for* `mult` x y.

Lemma 4.7.7. *Given a command c and term t, then:*

$$(\Theta c) * t = \Theta c$$

Proof. This follows directly from the rules $\to_{\mu\mathtt{N}}$ and $\to_{\mu R}$.

$$\begin{aligned}(\Theta c) * t &\equiv \mathtt{nrec}\ (\lambda y.0)\ (\lambda xhy.y + (hy))\ (\Theta c)\ t\\ &\to (\Theta c)t\\ &\to \Theta c\end{aligned}$$

□

The preceding lemma indicates that if the first argument of the multiplication function is of the shape Θc, it will propagate through the function. However, if the first argument is 0 and the second argument is of the shape Θc, then the function returns 0 and ignores the argument Θc.

$$\begin{aligned}0 * (\Theta c) &\equiv \mathtt{nrec}\ (\lambda y.0)\ (\lambda xhy.y + (hy))\ (\Theta c)\\ &\to (\lambda y.0)\,(\Theta c)\\ &\to 0\end{aligned}$$

This behavior is obviously what we would expect in a call-by-name system: if an argument is not used, it will be ignored.

Lemma 4.7.8. *Given a command c and a natural number $n > 0$, then:*

$$\underline{n} * (\Theta c) = \Theta c$$

Proof. This follows directly from Lemma 4.7.7.

$$\begin{aligned}\underline{n+1} * (\Theta c) &\equiv \mathtt{nrec}\ (\lambda y.0)\ (\lambda xhy.y + (hy))\ \underline{n+1}\ (\Theta c)\\ &\twoheadrightarrow (\Theta c) + (\underline{n} * (\Theta c))\\ &\twoheadrightarrow \Theta c\end{aligned}$$

□

Now that we have defined multiplication we are finally able to define our program and its specification.

Definition 4.7.9. *The term* Π *is defined as* $\Pi := \lambda f \lambda x.\mu\alpha.[\alpha]\Pi_f^\alpha\ x$*, where:*

$$\Pi_f^\alpha = \mathsf{Irec}\ \underline{1}\ (\lambda x\ m\ .\ \mathsf{ncase}\ (\Theta[\alpha]0)\ (\lambda y\ .\ \mathtt{S}y * m)\ (f\,x))$$

Definition 4.7.10. *Given a term* $f : \mathtt{N} \to \mathtt{N}$*, then the relation binary relation* M_f *over numerals is inductively defined as follows.*

$$M_f(\underline{0}, \underline{1})$$
$$\forall xm\ .\ M_f(x, m) \to M_f(\mathtt{S}x, f\,x * m)$$

We say that Π *is correct if* $M_f(\underline{x}, \Pi\, f\, \underline{x})$ *for each* $f : \mathtt{N} \to \mathtt{N}$ *and* $x \in \mathbb{N}$*.*

Keeping the distinction between returning normally and returning exceptionally in mind we introduce the following definition. However, soon we will see that this definition is too restricted.

Temporary Definition 4.7.11. *Given a term* $f : \mathtt{N} \to \mathtt{N}$*, a* μ*-variable* α *and a natural number* x*, then we say that:*

1. $\Pi_f^\alpha\ \underline{x}$ returns normally *if* $\Pi_f^\alpha\ \underline{x} = \underline{m}$ *with* $M_f(\underline{x}, \underline{m})$*.*
2. $\Pi_f^\alpha\ \underline{x}$ returns exceptionally *if* $\Pi_f^\alpha\ \underline{x} = \Theta[\alpha]\underline{m}$ *with* $M_f(\underline{x}, \underline{m})$*.*

Temporary Lemma 4.7.12. *Given a term* $f : \mathtt{N} \to \mathtt{N}$*, a* μ*-variable* α *and a natural number* x*, and let* $\Pi_f^\alpha\ \underline{x}$ *return normally or exceptionally, then we have* $M_f(\underline{x}, \Pi\, f\, \underline{x})$*.*

Proof. Suppose that $\Pi_f^\alpha\ \underline{x}$ returns normally. That is, $\Pi_f^\alpha\ \underline{x} = \underline{m}$ with $M_f(\underline{x}, \underline{m})$. Now we have:

$$\begin{aligned}
\Pi\, f\, \underline{x} &\equiv (\lambda f \lambda x.\mu\alpha.[\alpha]\Pi_f^\alpha\ x)\, f\, \underline{x} \\
&\to \mu\alpha.[\alpha]\Pi_f^\alpha\ \underline{x} \\
&= \mu\alpha.[\alpha]\underline{m}n \\
&\to \underline{m}
\end{aligned}$$

Alternatively, suppose that $\Pi_f^\alpha\ \underline{x}$ returns exceptionally. That is, $\Pi_f^\alpha\ \underline{x} = \Theta[\alpha]\underline{m}$ with $M_f(\underline{x}, \underline{m})$. Now we have:

$$\begin{aligned}
\Pi\, f\, \underline{x} &\equiv (\lambda f \lambda x.\mu\alpha.[\alpha]\Pi_f^\alpha\ x)\, f\, \underline{x} \\
&\to \mu\alpha.[\alpha]\Pi_f^\alpha\ \underline{x} \\
&= \mu\alpha.[\alpha]\Theta[\alpha]\underline{m} \\
&\to \mu\alpha.[\alpha]\underline{m} \\
&\to \underline{m}
\end{aligned}$$

□

Now it remains to prove that given a natural number x we have that $\Pi_f^\alpha\ \underline{x}$ returns normally or exceptionally. We proceed by induction on x. It follows immediately for $x = 0$, because then $\Pi_f^\alpha\ \underline{0} = \underline{1}$ and $M_f(\underline{0}, \underline{1})$. If we have $x > 0$, then we distinguish the following cases.

1. Let $f\,\underline{x} = \underline{0}$. Now we have $\Pi_f^\alpha\,\underline{x} = \Theta[\alpha]\underline{0}$. Hence we are done because we certainly have $M_f(\underline{x}, \underline{0})$.

2. Let $f\,\underline{x} \neq 0$. Now we have $\Pi_f^\alpha\,\underline{x} = f\,\underline{x} * (\Pi_f^\alpha\,\underline{x-1})$. Moreover, by the induction hypothesis we have either:

   (a) $\Pi_f^\alpha\,\underline{x-1}$ returns normally. That is $\Pi_f^\alpha\,\underline{x-1} = \underline{m}$ with $M_f(\underline{x-1}, \underline{m})$. Now $\Pi_f^\alpha\,\underline{x} = f\,\underline{x} * \underline{m}$, so we are done because $M_f(\underline{x}, f\,\underline{x} * \underline{m})$.

   (b) $\Pi_f^\alpha\,\underline{x-1}$ returns exceptionally. That is $\Pi_f^\alpha\,\underline{x-1} = \Theta[\alpha]\underline{m}$ with $M_f(\underline{x}, \underline{m})$. Now $\Pi_f^\alpha\,\underline{x} = f\,\underline{x} * \Theta[\alpha]\underline{m} = \Theta[\alpha]\underline{m}$ by Lemma 4.7.8. However, if $\underline{m}$ is the product of the sequence $(\underline{0}, \ldots, \underline{x})$, it is not necessarily the product of a longer sequence, so we are stuck.

Here we are stuck because the induction hypothesis is not strong enough. Because if we know that $\Pi_f^\alpha\,\underline{x-1}$ returns exceptionally, we cannot conclude that $\Pi_f^\alpha\,\underline{x}$ returns exceptionally.

But first, let us assume that we encounter a subcase $\Pi_f^\alpha\,\underline{x}$ for certain x during evaluation of $\Pi\,f\,\underline{n}$. Now, what does it mean, in terms of program correctness, to return exceptionally with $\underline{m}$? It means that $\underline{m}$ is a product of f applied to the sequence $(\underline{0}, \ldots, \underline{n})$ and not just $(\underline{0}, \ldots, \underline{x})$. More generally, it means that $\underline{m}$ is a product of f applied to every sequence $(\underline{0}, \ldots, \underline{y})$ such that $y > x$.

Definition 4.7.13. *Given a term $f : \mathtt{N} \to \mathtt{N}$, a μ-variable α and a natural number x, then we say that:*

1. $\Pi_f^\alpha\,\underline{x}$ returns normally *if* $\Pi_f^\alpha\,\underline{x} = \underline{m}$ *with* $M_f(\underline{x}, \underline{m})$.

2. $\Pi_f^\alpha\,\underline{x}$ returns exceptionally *if* $\Pi_f^\alpha\,\underline{x} = \Theta[\alpha]\underline{m}$ *with* $M_f(\underline{y}, \underline{m})$ *for each* $y \geq x$.

So if $\Pi_f^\alpha\,\underline{x}$ returns exceptionally, we do not just prove that $\underline{m}$ is the product of f applied to the sequence $(\underline{0}, \ldots, \underline{x})$, but also that $\underline{m}$ is the product of f applied to every sequence $(\underline{0}, \ldots, \underline{y})$ such that $y \geq x$.

Lemma 4.7.14. *Given a term $f : \mathtt{N} \to \mathtt{N}$, a μ-variable α and a natural number x. Also, let $\Pi_f^\alpha\,\underline{x}$ return normally or exceptionally, then we have $M_f(\underline{x}, \Pi\,f\,\underline{x})$.*

Proof. In the same way as we have proven Temporary Lemma 4.7.12, but take $y = x$ in case of an exceptional return. □

Lemma 4.7.15. *Given a term $f : \mathtt{N} \to \mathtt{N}$, a μ-variable α and a natural number x, then $\Pi_f^\alpha\,\underline{x}$ returns normally or exceptionally.*

Proof. By induction on x. It follows immediately for $x = 0$, because then $\Pi_f^\alpha\,\underline{0} = \underline{1}$ and $M_f(\underline{0}, \underline{1})$. If $x > 0$, then we distinguish the following cases.

1. Let $f\,\underline{x} = \underline{0}$. Now we have $\Pi_f^\alpha\,\underline{x} = \Theta[\alpha]\underline{0}$. Hence we are done because we have $M_f(\underline{y}, 0)$ for each $y \geq x$.

2. Let $f\,\underline{x} \neq 0$. Now we have $\Pi_f^\alpha\,\underline{x} = f\,\underline{x} * (\Pi_f^\alpha\,\underline{x-1})$. Moreover, by the induction hypothesis we have either:

   (a) $\Pi_f^\alpha\,\underline{x-1}$ returns normally. That is $\Pi_f^\alpha\,\underline{x-1} = \underline{m}$ with $M_f(\underline{x-1}, \underline{m})$. Now $\Pi_f^\alpha\,\underline{x} = f\,\underline{x} * \underline{m}$, so we are done because $M_f(\underline{x}, f\,\underline{x} * \underline{m})$.

(b) $\Pi_f^\alpha\, \underline{x-1}$ returns exceptionally. That is $\Pi_f^\alpha\, \underline{x-1} = \Theta[\alpha]\underline{m}$ with $M_f(\underline{y}, \underline{m})$ for each $y \geq x$. Now $\Pi_f^\alpha\, \underline{x} = f\, \underline{x} * \Theta[\alpha]\underline{m} = \Theta[\alpha]\underline{m}$ by Lemma 4.7.8. Hence $\Pi_f^\alpha\, \underline{x}$ returns exceptionally as well. □

Theorem 4.7.16. *The program* Π *is correct. That is,* $M_f(\underline{x}, \Pi\, f\, \underline{x})$ *for each* $f : \mathtt{N} \to \mathtt{N}$ *and* $x \in \mathbb{N}$.

Proof. By Lemma 4.7.14 and 4.7.15. □

It is not too hard to extend this approach to more interesting programs. It always consists of two parts: prove that `throw`s propagate through various terms and then define what it means to return normally respectively exceptionally.

Chapter 5

Further research

In this chapter we take a look at some extensions of our work. First we discuss extensions of $\lambda_\mu^{\mathbf{T}}$ with other data types. Specifically, we consider an extension with lists and indicate how the main meta theoretical properties can be proven. Unfortunately, our proof of strong normalization for $\lambda_\mu^{\mathbf{T}}$ does not extend to lists, this is possibly related to a known problem with respect to proving strong normalization for the symmetric λ_μ-calculus.

Secondly, we discuss how our system could be used for extraction of programs with control from classical proofs. In order to do this, we explain the concept of program extraction by considering program extraction from the Calculus of Constructions to System $\mathbf{F}_\omega$.

5.1 Other data types

In this section we extend $\lambda_\mu^{\mathbf{T}}$ with lists and indicate some problem of this extension. First we extend Definition 4.1.1 with a type $[\rho]$ for lists over a type ρ, Definition 4.2.1 with the list constructors `nil` and `cons` and with a primitive recursor `lrec` for lists as follows.

$$\rho, \delta ::= \ldots \mid [\delta]$$
$$t, r, s ::= \ldots \mid \texttt{nil} \mid \texttt{cons}\ r\ t \mid \texttt{lrec}_\rho\ r\ s\ t$$

As usual we denote $\texttt{cons}\ r\ t$ as $r :: t$. Moreover, because it is convenient to be able to talk about a term representing an actual list we introduce the following notation.

Notation 5.1.1. $[r_1, \ldots, r_n] := r_1 :: \ldots :: r_n :: \texttt{nil}$

The additional typing rules are shown in Figure 5.1. Now, to define the reduction rules, we have to be more careful. For $\lambda_\mu^{\mathbf{T}}$ we had to make sure that the reduction rule $\texttt{nrec}\ r\ s\ \texttt{S}t \to s\ t\ (\texttt{nrec}\ r\ s\ t)$ is only allowed if t is unable to reduce to a term of the shape $\mu\alpha.c$. Therefore we required t to be equal to $\underline{n}$ for some $n \in \mathbb{N}$. However, because this condition becomes slightly more complicated for lists, we extend the notion of values.

Definition 5.1.2. *The* values *of $\lambda_\mu^{\mathbf{T}}$ with list are inductively defined as follows.*

$$v, w ::= 0 \mid \texttt{S}v \mid \texttt{nil} \mid v :: w \mid \lambda x.r$$

$$\Gamma \vdash \mathtt{nil} : [\delta] \qquad \frac{\Gamma \vdash r : \delta \qquad \Gamma \vdash t : [\delta]}{\Gamma \vdash r :: t : [\delta]}$$

(a) nil (b) cons

$$\frac{\Gamma \vdash r : \rho \qquad \Gamma \vdash s : \delta \to [\delta] \to \rho \to \rho \qquad \Gamma \vdash t : [\delta]}{\Gamma \vdash \mathtt{lrec}_\rho\ r\ s\ t : \rho}$$

(c) lrec

Figure 5.1: The typing rules of $\lambda^{\mathbf{T}}_{\mu}$ extended with lists.

In order to extend the notion of structural substitution we have to extend contexts with some additional constructors first.

$$E ::= \ldots \mid E :: t \mid v :: E \mid \mathtt{lrec}\ r\ s\ E$$

Now the notion of substitution, structural substitution, free λ-variables, free μ-variables are extended in the obvious way and we can extend the reduction rules as follows.

$$\begin{array}{rcl}
\mathtt{lrec}\ r\ s\ \mathtt{nil} & \to_{\mathtt{nil}} & r \\
\mathtt{lrec}\ r\ s\ (v :: w) & \to_{::} & s\ v\ w\ (\mathtt{lrec}\ r\ s\ w) \\
\mathtt{lrec}\ r\ s\ (\mu\alpha.t) & \to_{\mu[]} & \mu\alpha.t[\alpha := \alpha\ (\mathtt{lrec}\ r\ s\ \Box)] \\
(\mu\alpha.t) :: r & \to_{\mu::L} & \mu\alpha.t[\alpha := \alpha\ (\Box :: r)] \\
v :: (\mu\alpha.t) & \to_{\mu::R} & \mu\alpha.t[\alpha := \alpha\ (v :: \Box)]
\end{array}$$

It is not too hard to verify that this system satisfies subject reduction and that we have a normal form theorem similar to Lemma 4.2.6. However we do not have a similar result for lists as we stated for numerals in Lemma 4.2.7. For example, $\mu\alpha.[\alpha][\lambda x.\Theta[\alpha][\lambda y.y]]$ is a well-typed term of type $[\gamma \to \gamma]$ and is moreover in normal form. But unfortunately this term is not of the shape $[r_1, \ldots, r_n]$. However, we have such a result for normal forms of *ground types*.

Definition 5.1.3. *A* ground type *is a type of the following shape.*

$$\rho, \delta ::= \mathtt{N} \mid [\delta]$$

Lemma 5.1.4. *Given a term t that is in normal form and such that $; \Delta \vdash t : \rho$, then:*

1. *If $\rho = \mathtt{N}$, then $t \equiv \underline{n}$ or $t \equiv \mu\alpha.[\beta]\underline{n}$ for some $n \in \mathbb{N}$.*
2. *If $\rho = [\delta]$, then $t \equiv [r_n, \ldots, r_1]$ or $t \equiv \mu\alpha.[\beta][r_n, \ldots, r_1]$ for some $n \in \mathbb{N}$ and terms r_i.*

   Furthermore, if δ is a ground type, then $\mathrm{FCV}(r_i) = \emptyset$ for $1 \leq i \leq n$.
3. *If $\rho = \gamma \to \delta$, then $t \equiv \lambda x.r$ or $t \equiv \mu\alpha.[\beta]\lambda x.r$ for some x and r.*

Proof. By induction on the derivation $; \Delta \vdash t : \rho$. We treat some cases.

(nil) Let $;\Delta \vdash \texttt{nil} : [\delta]$. Now we are immediately done because $\texttt{nil} \equiv [\,]$.

Furthermore, we have $\mathrm{FCV}(\texttt{nil}) = \emptyset$.

(cons) Let $;\Delta \vdash r_{n+1} :: t : [\delta]$ with $;\Delta \vdash r : \delta$ and $;\Delta \vdash t : [\delta]$. Now we have $t \equiv [r_n, \ldots, r_1]$ or $t \equiv \mu\alpha.[\beta][r_n, \ldots, r_1]$ by the induction hypothesis. In the former case we have $r :: t \equiv [r_{n+1}, \ldots, r_1]$. In the latter case we obtain a contradiction because the $\rightarrow_{\mu::L}$ or $\rightarrow_{\mu::R}$-rule can be applied.

Furthermore, if $[\delta]$ is a ground type, then we have $\mathrm{FCV}(r_{n+1}) = \emptyset$ and $\mathrm{FCV}(r_i) = \emptyset$ for $1 \leq i \leq n$ by the induction hypothesis. Hence we have $\mathrm{FCV}(r_i)$ for $1 \leq i \leq n+1$. □

Lemma 5.1.5. *Given a term t that is in normal form and such that $;\vdash t : \rho$ for a ground type ρ, then:*

1. *If $\rho = \mathtt{N}$, then $t \equiv \underline{n}$ for some $n \in \mathbb{N}$.*
2. *If $\rho = [\delta]$, then $t \equiv [r_1, \ldots, r_n]$ for some $n \in \mathbb{N}$ and terms r_i*

Proof. This result follows immediately from Lemma 5.1.4. □

5.1.1 Confluence

In order to prove confluence we extend the parallel reduction relation (Definition 4.3.2). For terms we add the clauses that are listed below, nothing changes for commands and for contexts we add the obvious clauses.

(t10) $\texttt{nil} \Rightarrow \texttt{nil}$

(t11) If $r \Rightarrow r'$, then $\texttt{lrec}\ r\ s\ \texttt{nil} \Rightarrow r'$.

(t12) If $r \Rightarrow r'$, $s \Rightarrow s'$, $v \Rightarrow v'$ and $w \Rightarrow w'$, then $\texttt{lrec}\ r\ s\ (v :: w) \Rightarrow s'\ v'\ w'\ (\texttt{lrec}\ r'\ s'\ w')$.

The following lemma is required to prove a substitution lemma (similar to Lemma 4.3.7) and structural substitution lemma (similar to Lemma 4.3.8).

Lemma 5.1.6. *For each value v we have:*

1. *$v[x := t]$ is a value for all variables x and terms t*
2. *$v[\alpha := \beta E]$ is a value for all μ-variables α and β and all contexts E.*

Proof. By induction on v. The only interesting cases are variables and commands. However, variables and commands are not values, so the required result follows immediately. □

Because various rules require that certain subterms are values, we need that values are preserved under reduction.

Lemma 5.1.7. *Given a value v such that $v \Rightarrow t$, then t is a value too.*

Proof. By induction on $v \Rightarrow t$. □

Now we can extend the classification of terms (Lemma 4.3.9) and the complete development (Definition 4.3.12) in a straightforward way.

1. $(\texttt{lrec}\ r\ s\ \texttt{nil})^\diamond := r^\diamond$
2. $(\texttt{lrec}\ r\ s\ (v :: w))^\diamond := s^\diamond\ v^\diamond\ w^\diamond\ (\texttt{lrec}\ r^\diamond\ s^\diamond\ w^\diamond)$
3. $(\texttt{lrec}\ r\ s\ u)^\diamond := \texttt{lrec}\ r^\diamond\ s^\diamond\ u^\diamond$
   provided that $u \not\equiv E[\mu\beta.c]$ and $u \not\equiv v :: w$
4. $(u :: t)^\diamond := u^\diamond :: v^\diamond$
   provided that $u \not\equiv E[\mu\beta.c]$ and $t \not\equiv F[\mu\gamma.d]$

We expect that all our results described in Section 4.3 can easily be repeated for the extension with lists.

5.1.2 Strong normalization

Unfortunately our proof of strong normalization does not extend to lists. Let us extend the definition of the set of reducibility candidates (Definition 4.4.14) in a straightforward way.

1. If $T \in \mathcal{R}$, then $\{\Box :: t \mid t \in T\} \to T \in \mathcal{R}$.
2. If $T, S \in \mathcal{R}$, then $\{v :: \Box \mid v \in S\} \to T \in \mathcal{R}$.
3. If $R, S, T \in \mathcal{R}$, then $\{\texttt{lrec}\ r\ s\ \Box \mid r \in T, s \in R \to S \to T \to T\} \to T \in \mathcal{R}$.

Definition 5.1.8. *Given a set of terms R, let $\mathcal{L}_R$ denote the smallest collection of terms satisfying the following conditions.*

1. $\texttt{SN} \in \mathcal{L}_R$
2. *If $S \in \mathcal{L}_R$, then $\{v :: \Box \mid v \in R\} \to S \in \mathcal{L}_R$.*
3. *If $S \in \mathcal{L}_R$ and $T \in \mathcal{R}$, then:*

$$\{\texttt{lrec}\ r\ s\ \Box \mid r \in T, s \in S \to T \to T\} \to T \in \mathcal{L}_R$$

Definition 5.1.9. *The* interpretation $[\![\rho]\!]$ *of a type ρ is defined as follows.*

$$[\![\mathtt{N}]\!] := \bigcap \mathcal{N}$$
$$[\![[\delta]]\!] := \bigcap \mathcal{L}_{[\![\delta]\!]}$$
$$[\![\delta \to \sigma]\!] := [\![\delta]\!] \to [\![\sigma]\!]$$

However, now we are stuck, because if we have $t \in [\![[\delta]]\!]$ and $s \in [\![\delta]\!]$, then we only have $s :: t \in [\![[\delta]]\!]$ in case s is a value. A first intuition tells us to consider contexts of the shape $t :: E$ instead of $v :: E$. Despite this results in a loss of confluence, strong normalization of this system implies strong normalization of our extension with lists. Clause (2) of Definition 5.1.8 becomes:

2. If $S \in \mathcal{L}_R$, then $\{t :: \Box \mid t \in R\} \to S \in \mathcal{L}_R$.

Unfortunately, this modification introduces many other problems. For example, our proof of Lemma 4.4.16 fails. Here we have to prove that $E[x] \in T$ for all $T \in \mathcal{R}$ and $E \in \mathtt{SN}^{\Box}$. If we proceed by induction on the generation of T, the first case requires us to prove that $E[x] \in \mathtt{SN}$. However, this is non-trivial. For example, let $E = \mu\alpha.c :: \Box$, then $\mu\alpha.c :: x \to \mu\alpha.c[\alpha := \alpha\ (\Box :: x)]$. Although certainly $c \in \mathtt{SN}$, it does not directly imply that $c[\alpha := \alpha\ (\Box :: x)] \in \mathtt{SN}$.

We expect that this problem is closely related to the problems described in [DN05]. In their work they explain a similar problem for the symmetric λ_μ-calculus and claim that the usual technique of reducibility candidates does not work. We moreover expect that we will encounter similar problems for an extension with other data types, for example, products.

5.2 Program extraction

Now that we have developed a system with both a control mechanism and data types one might wonder whether we could extend it with dependent types and then use it to extract programs with control from classical proofs. Before we go into detail we explain program extraction from the Calculus of Constructions so as to indicate what we precisely wish to achieve.

5.2.1 From the Calculus of Constructions

In this section we will briefly introduce program extraction from the Calculus of Constructions, which is a higher-order λ-calculus with dependent types, to System $\mathbf{F}_\omega$. Just as in [PM89], we consider three universes:

1. Data (data): this universe contains the types of System $\mathbf{F}_\omega$, which is the part of the of the Calculus on Constructions without dependent types. The types in this universe can be seen as the types of "ordinary" programs.

2. Specifications (spec): this universe contains propositions that have computationally relevant parts. These propositions are used to state specifications of programs in the data universe.

3. Propositions (prop): this universe contains propositions that have only logical content and hence no computationally relevant content. These propositions are used to state properties of programs in the data universe.

If we use the system merely for correctness proofs, then we use the first universe to write our program. Let us say that we have written the predecessor function $\mathtt{pred} : \mathsf{N} \to \mathsf{N}$, then we have $\mathsf{N} \to \mathsf{N} \in \mathsf{data}$. Now, the following formula states that $\mathtt{pred}$ is a correct predecessor function.

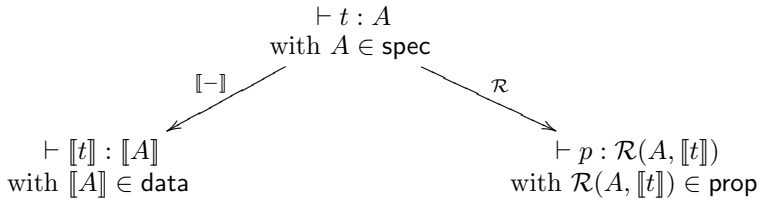
$$\mathtt{pred\_correct} := \forall n.(n = 0 \to \mathtt{pred}\, n = 0) \wedge (n > 0 \to \mathtt{pred}\, n + 1 = n)$$

Because $\mathtt{pred\_correct}$ is just a proposition without any computationally relevant content we have $\mathtt{pred\_correct} \in \mathsf{prop}$. Conversely, if we use the system for program extraction we would have written the following specification of the predecessor function.

$$\mathtt{pred\_spec} := \forall n.\exists m.(n = 0 \to m = 0) \wedge (n > 0 \to m + 1 = n)$$

Here we have `pred_spec` $\in$ `spec`, because we are going to use a proof of this proposition to extract a program that computes the predecessor.

More generally, let us consider a specification $A \in \mathsf{spec}$ and a proof t of A. Now, program extraction, as defined by Paulin [PM89], consists of an extraction map $[\![-]\!]$ and realizability map $\mathcal{R}$ such that:



Here, $[\![A]\!]$ is the type of the extracted program $[\![t]\!]$ and $\mathcal{R}(A, [\![t]\!]) \in \mathsf{prop}$ is a proposition which states that $[\![t]\!]$ is correct with respect to its specification A. So if we consider a proof t of `pred_spec`, then $[\![A]\!] = \mathsf{N} \to \mathsf{N}$ and $\mathcal{R}(A, [\![t]\!]) = \forall n.(n = 0 \to [\![t]\!]n = 0) \wedge (n > 0 \to [\![t]\!]n + 1 = n)$.

Unfortunately, System $\mathbf{F}_\omega$ is not quite suitable as a general purpose programming language because of its poor efficiency. For example, in System $\mathbf{F}_\omega$ one cannot define a predecessor function (on Church numerals) that reduces in constant time. Thus, having a System $\mathbf{F}_\omega$ program that is guaranteed to be correct is practically next to useless.

However, Letouzey [Let04] has extended Paulin's work to the Calculus of Inductive Constructions. This system supports inductive types and is therefore much closer to actual programming languages. Moreover, because of Letouzey's work, Coq is currently able to extract OCaml and Haskell programs that are guaranteed correct with respect to their specification. Also, Coq and Letouzey's work do not distinguish the universes data and spec, both universes are contained in the universe set.

5.2.2 From classical proofs

Repeating the methodology described in the previous section for extraction of programs with control from classical proofs presents the following challenges.

1. Development of a system with dependent types that is powerful enough to describe specifications of a desired class of programs.
2. Definition of an extraction map from this dependently typed system to a suitable control calculus without dependent types.
3. Definition of a classical realizability interpretation for this extraction map.

One should be very careful with respect to the first point, because this system should support just a limited amount of classical reasoning. It should surely not be able to prove an informative version of excluded middle $\forall A : \mathsf{spec}\,.\, A \vee \neg A$. If excluded middle were provable and moreover extracts to the type $1 + 1$, then the extracted program can decide provability in predicate logic, which is undecidable. Hence a logic that proves excluded middle is too strong.

Let us consider a variant of λ_μ for classical program extraction. Here we let occurrences of μ in the dependently typed system extract to occurrences of

μ in the target system. Now one could limit the amount of classical reasoning by ensuring that within a computationally irrelevant goal it is impossible to resume proving a computationally relevant goal. We would then restrict the activate/passivate rule as follows.

$$\frac{\Gamma; \Delta, \alpha : A \vdash M : B \qquad \beta : B \in (\Delta, \alpha : A) \qquad A \in \mathsf{spec} \iff B \in \mathsf{spec}}{\Gamma; \Delta \vdash \mu\alpha : A.[\beta]M : A}$$

This rule disqualifies the following proof of excluded middle, which by removing computationally irrelevant parts extracts to $\mu\alpha.[\alpha]\mathtt{in}_r()$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x : A; \alpha : A \vee \neg A \vdash x : A}{x : A; \alpha : A \vee \neg A \vdash \mathtt{in}_l x : A \vee \neg A}}{x : A; \alpha : A \vee \neg A \vdash \Theta[\alpha]\mathtt{in}_l x : \bot}}{; \alpha : A \vee \neg A \vdash \lambda x.\Theta[\alpha]\mathtt{in}_l x : \neg A}}{; \vdash \mu\alpha.[\alpha]\mathtt{in}_r(\lambda x.\Theta[\alpha]\mathtt{in}_l x) : A \vee \neg A}$$

Here we have $\bot \in \mathsf{prop}$, so, it is not allowed to resume proving $A \vee \neg A \in \mathsf{spec}$. Let us also consider a proof of the formula $\exists x.\mathsf{S}0 = x$ so as to illustrate that such a condition on the activate/passivate rule is necessary.

$$\dfrac{\dfrac{\dfrac{R : \mathsf{S}0 = \mathsf{S}0}{\langle \mathsf{S}0, R\rangle : \exists x.\mathsf{S}0 = x}}{\Theta[\alpha]\langle \mathsf{S}0, R\rangle : \mathsf{S}0 = 0}}{\mu\alpha.[\alpha]\langle 0, \Theta[\alpha]\langle \mathsf{S}0, R\rangle\rangle : \exists x.\mathsf{S}0 = x}$$

By removing computationally irrelevant parts we obtain the program $\mu\alpha.[\alpha]0$, which reduces to 0 and is therefore obviously not correct. In this proof, we have given an incorrect witness and while proving $\mathsf{S}0 = 0 \in \mathsf{prop}$, we have resumed proving the goal $\exists x.\mathsf{S}0 = x \in \mathsf{spec}$ so as to specify a correct witness. However, the proof of $\mathsf{S}0 = 0 \in \mathsf{prop}$, which contains our revised witness, is removed because it is computationally irrelevant.

Secondly, one should be careful that extracted programs of ground types actually reduce to a ground type. Hence it is important that the target system satisfies a suitable normal form theorem. For example, let us consider the following fictional proof of the formula $\exists x.\mathsf{S}0 = x$.

$$\dfrac{\dfrac{\dfrac{\dfrac{R : \mathsf{S}0 = \mathsf{S}0}{\langle \mathsf{S}0, R\rangle : \exists x.\mathsf{S}0 = x}}{\Theta[\alpha].\langle \mathsf{S}0, R\rangle : \exists x.0 = x} \qquad \dfrac{\dfrac{P : \mathsf{S}0 = \mathsf{S}y}{\langle \mathsf{S}y, P\rangle : \exists x.\mathsf{S}0 = x}}{\lambda y\lambda k.\langle \mathsf{S}y, P\rangle : \forall y.0 = y \to \exists x.\mathsf{S}0 = x}}{\mathtt{dest}\,\Theta[\alpha].\langle \mathsf{S}0, R\rangle\,\mathtt{in}\,\lambda y\lambda k.\langle \mathsf{S}y, P\rangle : \exists x.\mathsf{S}0 = x}}{\mathtt{dest}\,\mu\alpha.[\alpha]\Theta[\alpha].\langle \mathsf{S}0, R\rangle\,\mathtt{in}\,\lambda y\lambda k.\langle \mathsf{S}y, P\rangle : \exists x.\mathsf{S}0 = x}$$

By removing computationally irrelevant parts from this proof we obtain the program $t \equiv \mu\alpha.[\alpha](\lambda y.\mathsf{S}y)\Theta[\alpha]\mathsf{S}0$. As we have seen, in the $\lambda^{\mathbf{2}}_{\mu}$-calculus, this program does not reduce to a Church numeral. Hence it can never be correct with respect to its specification.

We expect that it is straightforward to develop a dependently typed version of the $\lambda^{\mathbf{T}}_{\mu}$-calculus and an extraction map to $\lambda^{\mathbf{T}}_{\mu}$-calculus. However, in order to use it for specification of programs further work is required. Also, defining a realizability interpretation seems non-trivial.

Chapter 6

Conclusions

In this thesis we have investigated various control calculi and developed the $\lambda_\mu^{\mathbf{T}}$-calculus, a Gödel's $\mathbf{T}$ version of the λ_μ-calculus. We have proven that $\lambda_\mu^{\mathbf{T}}$ satisfies the main theoretical properties: confluence, subject reduction, a normal form theorem and strong normalization. Also, we have presented an embedding of $\lambda_\mu^{\mathbf{T}}$ into $\lambda^{\mathbf{T}}$ and $\lambda^{\mathbf{2}}$. The first indicates that adding control to Gödel's $\mathbf{T}$ does not extend the class of definable functions. The second contrasts the differences between data types in $\lambda_\mu^{\mathbf{T}}$ and $\lambda_\mu^{\mathbf{2}}$.

6.1 Comparison of the systems

The $\lambda_{\mathcal{C}}$-calculus is originally described by an evaluation strategy. However, a system based on an evaluation strategy is not a well-suited framework for reasoning about programs, because if we prove a certain property of a program t, then that property is not necesarrily preserved if we "plug" the program t into another program. Hence various people tried to develop reduction theories for $\lambda_{\mathcal{C}}$, but unfortunatelly, none of these theories is able to minic the behavior of the given evaluation strategy.

On the other hand we have investigated the λ_Δ-calculus, which is presented by a reduction theory and satisfies the main theoretical properties. However, it has a defect with respect to producing proper normal forms and is moreover not able to simulate `catch` and `throw` well.

Furthermore, we have investigated the λ_μ-calculus, a system which is quite similar to λ_Δ. Since λ_μ distinguishes ordinary variables from continuation variables and moreover distinguishes commands from terms it does not suffer from the defects of λ_Δ. Also, we have studied the $\lambda_\mu^{\mathbf{2}}$-calculus, a second-order variant of λ_μ. Similarly as $\lambda^{\mathbf{2}}$, this system is able to encode most data types. However, while it has a defect with respect to producing proper normal-forms, a proper normal form can be obtained by use of output operators.

Because the λ_μ-calculus turned out to be most suitable system of those we have studied, we have used it for the development of the $\lambda_\mu^{\mathbf{T}}$-calculus in the second part of this thesis.

6.2 Call-by-name or call-by-value

In order to maintain confluence and a normal form theorem we had to develop the reduction rules of the $\lambda_\mu^{\mathbf{T}}$-calculus with care. Some of its reduction rules are closer to a call-by-value system than to the ones one would expect of a call-by-name system (which λ_μ originally is). Firstly, it contains the reduction rule $\mathtt{S}\mu\alpha.c \to \mu\alpha.c[\alpha := \alpha\ (\mathtt{S}\Box)]$, which is required to maintain a normal form theorem for numerals. Secondly, in order to unfold $\mathtt{nrec}\ r\ s\ (\mathtt{S}t)$ we have to reduce t to an actual numeral. This is required because it would otherwise result in a loss of confluence.

Our embedding of $\lambda_\mu^{\mathbf{T}}$ into $\lambda^{\mathbf{T}}$ justifies our particular choice of reduction rules. If we have a term $t^\circ : \rho^\circ$, then the only way to obtain a numeral from t° is by reducing it to a value first. So, for the case $(\mathtt{nrec}\ r\ s\ t)^\circ$, we have to reduce t° first in order to unfold the recursion. This is of course kind of obvious, because if we consider a numeral as a finite list of units, the only way to know whether an exception will be thrown, is to evaluate it to a value.

An important aspect of a call-by-name system is lazy evaluation of data types. However, because of these call-by-value like reduction rules we can conclude that a call-by-name system is not well suited for an extension with both data types and control mechanisms. Therefore it would be interesting to repeat the developments in Chapter 4.2 for a call-by-value system. We expect that all properties, except strong normalization, can be proven in a similar way. For a strong normalization proof by reducibility candidates we except similar problems as described in [DN05].

6.3 Primitive `catch` and `throw`

Instead of the λ_μ-calculus it would be interesting to consider a system with the control operators `catch` and `throw` as primitive. Such a system is described by Herbelin [Her10] where he uses it for an intuitionistic logic that proves a variant of Markov's principle.

$$\frac{\Gamma; \Delta, \alpha : \rho \vdash t : \rho}{\Gamma; \Delta \vdash \mathtt{catch}\ \alpha\ t : \rho} \qquad \frac{\Gamma; \Delta \vdash t : \rho \qquad \alpha : \rho \in \Delta}{\Gamma; \Delta \vdash \mathtt{throw}\ \alpha\ t : \delta}$$

(a) catch (b) throw

Having these operators as primitive has as advantage that the types in the context Δ can be restricted to a certain class. For example, in [Her10], the types in Δ should be $\forall$- and $\to$-free. As a result, one obtains a normal form theorem for function types too. In λ_μ one cannot enforce such a restriction on the activate rule because that leads to a loss of subject reduction. For example, for an arbitrary ρ, we have that $\mathtt{nrec}\ r\ s\ \mu\alpha.c$ with $\alpha : \mathtt{N}$ and $s : \rho$ reduces to $\mu\alpha.c[\alpha := \alpha\ (\mathtt{nrec}\ r\ s\ \Box)]$ with $\alpha : \rho$.

6.4 Implementation

One might wonder whether the $\lambda_\mu^{\mathbf{T}}$-calculus simulates any real implementations of control mechanisms. For example, the control mechanisms present in the function programming languages Scheme, Lisp or OCaml. First of all, most actual programming languages that support control are call-by-value while $\lambda_\mu^{\mathbf{T}}$ is call-by-name. Hence, as already noticed in Section 6.2, it would be a good idea to repeat our developments for a call-by-value system.

Moreover, the $\lambda_\mu^{\mathbf{T}}$-calculus supports statically bound exceptions instead of dynamically bound exceptions. Statically bound exceptions appear in the functional programming language Scheme while most other functional programming languages support dynamically bound exceptions. To illustrate the difference we consider the following Scheme program.

```
(call/cc (lambda (c)
  (+ 1 (
    (lambda (f) (call/cc (lambda (c) (f 0))))
    (lambda (x) (c x))
  ))
))
```

Here, both λ-binders bind different occurrences of the variable `c`, so instead of `call/cc (lambda (c) (f 0))` one could rename `c` into `d` and have written `call/cc (lambda (d) (f 0))`. Therefore, evaluation of `c x` results in a jump to the context captured by the outermost `call/cc` and so evaluation of the complete program yields 0.

However, if we consider a similar program in Lisp, which supports dynamically bound exceptions, we get another result.

```
(catch 'A
  (+ 1 (
    (lambda (f) (catch 'A (funcall f 0)))
    (lambda (x) (throw 'A x))
  ))
)
```

Here, evaluation of `throw 'A x` results in a jump to the innermost `catch`, so evaluation of the complete program yields 1. We can most likely modify our system to dynamically bound exceptions by considering substitution that is not capture avoiding for μ-variables. However, it is unclear whether our results are preserved for this modification.

Appendix A

Classical program extraction in Coq

In this appendix we illustrate, by considering the list product function, program extraction from a classical proof. Our approach is similar to that of Caldwell, Gent and Underwood, who considered program extraction from classical proofs in the proof assistant NuPrl [CGU00]. In their work they extended NuPrl with a proof rule for Peirce's law and associated `call/cc` for its extraction. Here we use the proof assistant Coq [CDT] in which we associate the control operator $\mathcal{C}$ for extraction of double negation[1].

Of course, this approach does not guarantee correctness of the extracted program. In fact, we will show that it is quite easy to construct proofs whose extraction is not correct with respect to its specification. However, it shows that certain classical proofs do contain computational content.

First we assume the following classical axioms.

```
Parameter ⫫ : set.
Parameter dn : ∀S : set . ((S → ⫫) → ⫫) → S.
Definition efq : ∀S : set . ⫫ → S := λSM . dn(λ_.M).
```

Note that instead of Coq's computationally irrelevant type `False` we use our own type ⫫ which has type set and is therefore computationally relevant. Next, we let the axioms `dn` and `efq` extract to the control operators $\mathcal{C}$ and $\mathcal{A}$, respectively.

```
Extract Inlined Constant dn => 𝒞.
Extract Inlined Constant efq => 𝒜.
```

In order to prove that each list has a product, we shall give a specification of the product of a list of natural numbers.

Definition A.1. *The binary relation M over lists of natural numbers and natural numbers is inductively defined as follows.*

$$m_{\mathsf{nil}} : M(\mathsf{nil}, 1) \qquad m_{\mathsf{cons}} : \forall l\, p\, x\, .\, M(l,p) \to M(x :: l, x * p)$$

[1] In Coq it is impossible to define new binders or introduce a special kind of variables, hence we are unable to extend it to the λ_μ-calculus.

Now, we say that a natural number n is the product of a list l *if* $M(l, n)$.

One should still convince oneself that we have indeed given a correct specification of the product of a list of natural numbers. To get more confidence in this specification we prove that it specifies the product uniquely.

Lemma A.2. *The relation M specifies the product of a list uniquely. That is:*

$$\forall l\, n_1\, n_2\, .\, M(l, n_1) \land M(l, n_2) \to n_1 = n_2$$

Since the specification corresponds to the graph of the obvious list product function, it is easy to give a constructive proof of $\forall l \exists n\, .\, M(l, n)$.

$$\dfrac{\dfrac{M(\mathsf{nil}, 1)}{\exists n.M(\mathsf{nil}, n)} \qquad \dfrac{\dfrac{\dfrac{M(x :: l, x * m)}{\exists n.M(x :: l, n)}}{\forall m.M(l, m) \to \exists n.M(x :: l, n)}}{\exists n.M(x :: l, n)}}{\forall l \exists n\, .\, M(l, n)}$$

Now, Coq extracts the following program from this proof.

```
let rec listmult l = match l with
  | nil    -> 1
  | x :: k -> x * (listmult k)
```

In order to obtain a program that uses the control operators $\mathcal{C}$ and $\mathcal{A}$ we will construct a classical proof. Intuitively one would be urged to use double negation directly so as to passivate the goal $\exists n.M(l, n)$ and resume proving it in the $0 :: l$ case. Such a proof would look as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{M(\mathsf{nil}, 1)}{\exists n.M(\mathsf{nil}, n)} \qquad \dfrac{\dfrac{\vdots}{\bot\!\!\!\bot}}{\exists n.M(0 :: l, n)}\,\mathtt{efq} \qquad \dfrac{\vdots}{x \neq 0 \to \exists n.M(x :: l, n)}}{\exists n.M(x :: l, n)}}{\exists n.M(l, n)}}{\dfrac{\dfrac{\bot\!\!\!\bot}{((\exists n.M(l, n)) \to \bot\!\!\!\bot) \to \bot\!\!\!\bot}\,\mathtt{dn}}{\exists n.M(l, n)}}{\forall l \exists n.M(l, n)}$$

Here, after we have used double negation, we proceed by induction on l and distinguish the cases $x = 0$ and $x \neq 0$. If $x = 0$, we use Ex Falso and apply $\exists n.M(l, n) \to \bot\!\!\!\bot$ so as to resume our passivated goal.

Unfortunately, program extraction of this proof does not yield the expected program, because induction on l also affects the list l in the context. The program that Coq extracts looks as follows.

```
let listmult2 l_tp = C(λh_tp.h_tp
  (let rec listmult2_core l h = match l with
   | nil    -> 1
   | x :: k -> if (x = 0)
               then A(h 0)
               else x * (listmult2_core k (λg.h(x * g)))
   in listmult2_core l_tp h_tp))
```

Now, whenever a zero is encountered the continuation h is invoked. However, since this continuation is modified each iteration, this is not desirable. Instead, we would like it to invoke the continuation h_{tp} and break out of the recursion. To illustrate what happens, we apply this program to the list $[4,3,0,1]$. We abbreviate `listmult2` as `lm` and `listmult2_core` as `lmc`.

$$
\begin{aligned}
E[\,\texttt{lm}\ [4,3,0,1]\,] &= E[\,\mathcal{C}(\lambda h.h\ (\texttt{lmc}\ [4,3,0,1]\ h))\,]\\
&\rhd (\lambda h.h\ (\texttt{lmc}\ [4,3,0,1]\ h))\lambda x.\mathcal{A}E[x]\\
&\rhd (\lambda x.\mathcal{A}E[x])(\texttt{lmc}\ [4,3,0,1]\ \lambda x.\mathcal{A}E[x])\\
&\rhd \mathcal{A}E[\,\texttt{lmc}\ [4,3,0,1]\ \lambda x.\mathcal{A}E[x]\,]\\
&\rhd E[\,\texttt{lmc}\ [4,3,0,1]\ \lambda x.\mathcal{A}E[x]\,]\\
&\rhd E[\,4*(\texttt{lmc}\ [3,0,1]\ \lambda g.(\lambda x.\mathcal{A}E[x])\ (4*g))\,]\\
&\rhd E[\,4*3*(\texttt{lmc}\ [0,1]\ \lambda j.(\lambda g.(\lambda x.\mathcal{A}E[x])\ (4*g))\ (3*j))\,]\\
&\rhd E[\,4*3*\mathcal{A}((\lambda j.(\lambda g.(\lambda x.\mathcal{A}E[x])\ (4*g))\ (3*j))\ 0)\,]\\
&\rhd E[\,(\lambda j.(\lambda g.(\lambda x.\mathcal{A}E[x])\ (4*g))\ (3*j))\ 0\,]\\
&\rhd E[\,(\lambda g.(\lambda x.\mathcal{A}E[x])\ (4*g))\ (3*0)\,]\\
&\rhd E[\,(\lambda g.(\lambda x.\mathcal{A}E[x])\ (4*g))\ 0\,]\\
&\rhd E[\,(\lambda x.\mathcal{A}E[x])\ (4*0)\,]\\
&\rhd E[\,(\lambda x.\mathcal{A}E[x])\ 0\,]\\
&\rhd E[\,\mathcal{A}E[0]\,]\\
&\rhd E[\,0\,]
\end{aligned}
$$

As this evaluation sequence shows, not only is an element multiplied by all its previous values when the function returns normally, but also when the continuation is invoked. Notice that this program is basically a more obscure version of the following program.

```
let rec listmult l = match l with
  | nil    -> 1
  | 0 :: k -> 0
  | x :: k -> x * (listmult k)
```

Before we continue, let us think why it is actually allowed to jump out when a zero is encountered? Maybe we should rephrase our proof such that the following lemma is used.

Lemma A.3. $m_0 : \forall l\,.\,0 \in l \to M(l,0)$

Furthermore, we do not want that l in the assumption $((\exists n.M(l,n)) \to \bot\!\!\!\bot$ is affected by induction. In order to keep that list separately, we prove the

following auxiliary theorem first.

$$\ldots, (\exists n\,.\,M(k,n)) \to \bot\!\!\!\bot \vdash \forall l\,.\,E(k,l) \to \exists n\,.\,M(l,n)$$

Here, the relation E is defined as follows.

Definition A.4. *The binary relation E over lists is inductively defined as follows.*

$$e_{\mathsf{base}} : \forall l\,.\,E(l,l) \qquad e_{\mathsf{cons}} : \forall l_1\,l_2\,x\,.\,E(l_1,l_2) \to E(x :: l_1, l_2)$$

Now, we say that l ends with *k if $E(l,k)$.*

This generalization is very similar to the methodology we have used to prove the correctness of the program consider in Section 4.7. In this auxiliary theorem, one should think of the assumption $(\exists n\,.\,M(k,n)) \to \bot\!\!\!\bot$ as as a passivated goal in the λ_μ-calculus.

$$\ldots; \exists n\,.\,M(k,n) \vdash \forall l\,.\,E(k,l) \to \exists n\,.\,M(l,n)$$

To prove the required theorem we state some obvious lemmas first.

Lemma A.5. $e_{\mathtt{inv}} : \forall l_1\,l_2\,x\,.\,E(l_1, x :: l_2) \to E(l_1,l_2)$

Lemma A.6. $e_{\mathtt{in}} : \forall l_1\,l_2\,x\,.\,E(l_1, x :: l_2) \to x \in l_1$

Using these lemmas, we are finally able to prove the required auxiliary theorem. The proof is shown below.

$$\dfrac{\dfrac{M(\mathsf{nil},1)}{\exists n.M(\mathsf{nil},n)}}{E(k,\mathsf{nil}) \to \exists n.M(\mathsf{nil},n)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{E(k,0::l)}{0 \in k}\,\text{A.6}}{M(k,0)}\,\text{A.3}}{\exists n.M(k,n)}}{\bot\!\!\!\bot}}{\exists n.M(0::l,n)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{E(k,x::l)}{E(k,l)}\,\text{A.5}}{E(k,l)} \qquad \dfrac{M(l,m)}{M(x::l, x*m)}}{\forall m.M(l,m) \to \exists n.M(x::l,n)}}{\exists n.M(x::l,n)}}{x \neq 0 \to \exists n.M(x::l,n)}$$

$$\dfrac{E(k,\mathsf{nil}) \to \exists n.M(\mathsf{nil},n) \qquad \exists n.M(x::l,n)}{E(k,x::l) \to \exists n.M(x::l,n)}$$

$$\forall l\,.\,E(k,l) \to \exists n\,.\,M(l,n)$$

Now, by letting $l = k$, it is trivial to prove the actual theorem.

$$\dfrac{\dfrac{\dfrac{\dfrac{E(l,l)}{\exists n.M(l,n)}}{\bot\!\!\!\bot}}{((\exists n.M(l,n)) \to \bot\!\!\!\bot) \to \bot\!\!\!\bot}\,\mathtt{dn}}{\exists n.M(l,n)}$$

$$\forall l \exists n.M(l,n)$$

Program extraction of this proof results in the expected program.

```
let listmult3 l = C(λh.h
  (let rec listmult3_core l = match l with
    | nil -> 1
    | x :: k -> if (x = 0)
                then A(h0)
                else x * (listmult3_core k)
  in listmult3_core l))
```

Applying this program to the list $[4,3,0,1]$ results in the following evaluation sequence. We abbreviate `listmult3` as `lm` and `listmult3_core` as `lmc`.

$$\begin{aligned}
E[\,\texttt{lm}\ [4,3,0,1]\,] &= E[\,\mathcal{C}(\lambda h.h\ (\texttt{lmc}\ [4,3,0,1]))\,] \\
&\rhd (\lambda h.h\,(\texttt{lmc}\ [4,3,0,1]))\ \lambda x.\mathcal{A}E[x] \\
&\rhd (\lambda x.\mathcal{A}E[x])\ (\texttt{lmc}\ [4,3,0,1]) \\
&\rhd \mathcal{A}E[\,\texttt{lmc}\ [4,3,0,1]\,] \\
&\rhd E[\,\texttt{lmc}\ [4,3,0,1]\,] \\
&\rhd E[\,4 * (\texttt{lmc}\ [3,0,1])\,] \\
&\rhd E[\,4 * 3 * (\texttt{lmc}\ [0,1])\,] \\
&\rhd E[\,4 * 3 * \mathcal{A}((\lambda x.\mathcal{A}E[x])\ 0)\,] \\
&\rhd (\lambda x.\mathcal{A}E[x])\ 0 \\
&\rhd \mathcal{A}E[\,0\,] \\
&\rhd E[\,0\,]
\end{aligned}$$

As we have noticed before, this example merely indicates that some classical proofs contain computational content. It does not guarantee that the extracted program is correct with respect to its specification. Such incorrect programs can be obtained by abusing the pitfalls described in Section 5.2.2. For example, we can resume a computationally relevant goal while proving a computationally irrelevant goal.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{0=0}{\exists x.x=0}}{\bot\!\!\!\bot}}{2=0}}{\exists x.x=0}}{\bot\!\!\!\bot}}{\exists x.x=0}$$

Here, while we are proving the goal $2 = 0 \in \mathsf{prop}$, we resume proving the goal $\exists x.x = 0 \in \mathsf{set}$ so as to revise our previously given witness. Now by removing computationally irrelevant parts program extraction results in the program $\mathcal{C}(\lambda k.k\,2)$. This program evaluates to 2, so it is obviously not correct.

Bibliography

[AH03] Zena M. Ariola and Hugo Herbelin. Minimal Classical Logic and Control Operators. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP*, volume 2719 of *LNCS*, pages 871–885. Springer, 2003.

[AH08] Zena M. Ariola and Hugo Herbelin. Control Reduction Theories: the Benefit of Structural Substitution. *Journal of Functional Programming*, 18(3):373–419, 2008.

[Bar92] Henk Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

[BB85] Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed Λ-programs on term algebras. *Theoretical Computer Science*, 39:135 – 154, 1985.

[BBS00] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined Program Extraction from Classical Proofs. In *Annals of Pure and Applied Logic*, pages 77–97. Springer Verlag, 2000.

[BHF01] Kensuke Baba, Sachio Hirokawa, and Ken-etsu Fujita. Parallel Reduction in Type Free λ_μ-calculus. *Electronic Notes in Theoretical Computer Science*, 42, 2001.

[BU02] Gilles Barthe and Tarmo Uustalu. CPS Translating Inductive and Coinductive Types. In Peter Thiemann, editor, *PEPM*, pages 131–142. ACM, 2002.

[CDT] The Coq Development Team. The Coq Proof Assistant. Web page, obtained from `http://coq.inria.fr` on December 5, 2009.

[CF98] Loïc Colson and Daniel Fredholm. System T, call-by-value and the minimum problem. *Theoretical Computer Science*, 206(1-2):301 – 315, 1998.

[CGU00] James L. Caldwell, Ian P. Gent, and Judith Underwood. Search Algorithms in Type Theory. *Theoretical Computer Science*, 232(1-2):55–90, 2000.

[CP09] Tristan Crolard and Emmanuel Polonowski. A program logic for higher-order procedural variables and non-local jumps, 2009. Obtained from `http://lacl.univ-paris12.fr/crolard/publications/Jumps.pdf` on May 7, 2010.

[dG94] Philippe de Groote. A CPS-translation of the λ_μ-calculus. In Sophie Tison, editor, *CAAP*, volume 787 of *LNCS*, pages 85–99. Springer, 1994.

[DN05] René David and Karim Nour. Why the usual candidates of reducibility do not work for the symmetric λ_μ-calculus. *Electronic Notes in Theoretical Computer Science*, 140:101–111, 2005.

[FF86] Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In Martin Wirsing, editor, *3rd Working Conference on the Formal Description of Programming Concepts*, pages 193–219. North-Holland Publishing, 1986.

[FFKD87] Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[FH92] Matthias Felleisen and Robert Hieb. The Revised Report on the Syntactic Theories of Sequential Control and State. *Theoretical Computer Science*, 103(2):235–271, 1992.

[Fri78] Harvey Friedman. Classically and intuitionistically provably recursive functions. In Gert Mller and Dana Scott, editors, *Higher Set Theory*, volume 669 of *Lecture Notes in Mathematics*, pages 21–28. Springer Verslag, 1978.

[Fuj97] Ken-etsu Fujita. Calculus of Classical Proofs I. In R. K. Shyamasundar and Kazunori Ueda, editors, *ASIAN*, volume 1345 of *LNCS*, pages 321–335. Springer, 1997.

[Fuj99] Ken-etsu Fujita. Explicitly Typed λ_μ-calculus for Polymorphism and Call-by-Value. In Jean-Yves Girard, editor, *TLCA*, volume 1581 of *LNCS*, pages 162–176. Springer, 1999.

[Fuj03] Ken-etsu Fujita. A Sound and Complete CPS-Translation for lambda-mu-Calculus. In Martin Hofmann, editor, *TLCA*, volume 2701 of *LNCS*, pages 120–134. Springer, 2003.

[Geu08] Herman Geuvers. Introduction to type theory. In Ana Bove, Luís Soares Barbosa, Alberto Pardo, and Jorge Sousa Pinto, editors, *LerNet ALFA Summer School*, volume 5520 of *LNCS*, pages 1–56. Springer, 2008.

[Goo75] John B. Goodenough. Structured exception handling. In *POPL*, pages 204–224. ACM, 1975.

[Gri90] Timothy G. Griffin. A Formulae-as-Types Notion of Control. In *POPL*, pages 47–58. ACM, 1990.

[GTL89] Jean Y. Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, 1989.

[Her10] Hugo Herbelin. An intuitionistic logic that proves markov's principle. In *LICS*, pages 50–56. IEEE Computer Society, 2010.

[IN06] Satoshi Ikeda and Koji Nakazawa. Strong normalization proofs by CPS-translations. *Information Processing Letters*, 99(4):163–170, 2006.

[Kre58] Georg Kreisel. Mathematical Significance of Consistency Proofs. *Journal of Symbolic Logic*, 23(2):155–182, 1958.

[Lau93] John Launchbury. A Natural Semantics for Lazy Evaluation. In *POPL*, pages 144–154. ACM, 1993.

[Let04] Pierre Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, 2004.

[Mur90] Chetan Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Cornell University, 1990.

[Nak03] Koji Nakazawa. Confluency and Strong Normalizability of Call-by-Value λ_μ-calculus. *Theoretical Computer Science*, 290(1):429–463, 2003.

[Par92] Michel Parigot. λ_μ-calculus: An Algorithmic Interpretation of Classical Natural Deduction. In Andrei Voronkov, editor, *LPAR*, volume 624 of *LNCS*, pages 190–201. Springer, 1992.

[Par93] Michel Parigot. Classical Proofs as Programs. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Kurt Gödel Colloquium*, volume 713 of *LNCS*, pages 263–276. Springer, 1993.

[Par97] Michel Parigot. Proofs of Strong Normalisation for Second Order Classical Natural Deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

[PM89] Christine Paulin-Mohring. Extracting F_ω's programs from proofs in the Calculus of Constructions. In *POPL*. ACM, 1989.

[RS94] Jakob Rehof and Morten Heine Sørensen. The λ_Δ-calculus. In Masami Hagiya and John C. Mitchell, editors, *TACS*, volume 789 of *LNCS*, pages 516–542. Springer, 1994.

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and The Foundations of Mathematics*. Elsevier Science, 2006.

[Tai67] William W. Tait. Intensional Interpretations of Functionals of Finite Type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tak95] Masako Takahashi. Parallel Reductions in λ-Calculus. *Information and Computation*, 118(1):120–127, 1995.

Index