$\lambda\mu$PRL – A Proof Refinement Calculus for Classical Reasoning in Computational Type Theory

Diplomarbeit

von

Nuria Brede

Lehrstuhl für Theoretische Informatik
Institut für Informatik
Universität Potsdam

Betreuung:
Prof. Dr. Christoph Kreitz
Dipl. Inform. Martin Gebser

27. Januar 2009

Abstract

This thesis is concerned with an integration of classical logic into computational type theory. Different from other approaches, our primary objective is not the extraction of constructive content from classical proofs, but rather the development of a hybrid proof environment which enables the user to benefit from the advantages of both concepts.

Computational Type Theory – as implemented in the Nuprl proof development system [CAB$^+$86] – provides the means to extract functional programs from the constructive proof of their specification. These programs are *correct by construction* because of the well-known *proofs-as-programs* correspondence.

However, finding proofs in the underlying constructive logic is harder than in classical logic. At the same time a proof might contain subgoals which do not even require a construction. But if the system was hybrid, the user could decide which logic to use in particular parts of the proof.

Still, it must be possible to distinguish between classical and constructive proofs to maintain the soundness of the system. Such a possibility is given, if the use of classical logic is subsequently visible in the proof term. A calculus providing this feature is the $\lambda\mu\nu$-calculus by Pym and Ritter [PR04]. Thus we develop the calculus $\lambda\mu$PRL, combining $\lambda\mu\nu$ with a top-down sequent calculus closely related to Nuprl's logic.

Furthermore, we consider $\lambda\mu$PRL under different aspects. We establish a confluent reduction relation and examine how the use of classical reasoning is indicated in a $\lambda\mu$PRL-term. We show that it is possible to make fine distinctions between constructive and non-constructive proofs (instead of dismissing every term containing one of the classical constructs). In this context we adapt the notion of *μ-safe* terms introduced by Crolard [Cro02], and propose an even more precise extension of this concept.

Finally we relate $\lambda\mu$PRL to the calculus $PROG_K$ for Peano arithmetic by Murthy [Mur91b]. Murthy's influential work was also situated in the Nuprl context and concerned the extraction of constructive content from classical proofs of arithmetical Σ^0_1 sentences. We show that a variant of $\lambda\mu$PRL is isomorphic to $PROG_K$.

Zusammenfassung

In der vorliegenden Arbeit beschäftigen wir uns mit der Möglichkeit klassischen Schließens in der konstruktiven Typentheorie. Dabei zielen wir jedoch weniger auf den konstruktiven Inhalt klassischer Beweise als auf eine hybride Beweisumgebung ab. Diese soll dem Benutzer ermöglichen, frei zwischen klassischen und konstruktiven Schließen zu wählen und somit die Vorteile beider Denkweisen nutzen zu können.

Die von uns betrachtete konstruktive Typentheorie ist im Nuprl Beweisentwicklungs-System [CAB$^+$86] implementiert. Nuprl bietet die Möglichkeit, funktionale Programme aus dem konstruktiven Beweis ihrer Spezifikation zu extrahieren. Aus dem bekannten *Beweise-als-Programme*Prinzip ergibt sich, dass solche Programme per Konstruktion korrekt sind. Klassische Logik dagegen lässt Widerspruchsbeweise zu, die nicht notwendigerweise konstruktiven Inhalt besitzen.

Allerdings ist die Beweissuche in konstruktiver Logik aufwendiger als in klassischer Logik. Des Weiteren kann ein Beweis Teilziele enthalten, die ohnehin keine Konstruktion erfordern. Als Lösung bietet sich ein hybrides System an: In diesem könnte der Benutzer entscheiden, welche Logik er in verschiedenen Beweisteilen einsetzen möchte.

Um die Korrektheit eines solchen Systems zu gewährleisten, muss jedoch die Möglichkeit gegeben sein, zwischen konstruktiven und klassischen Beweisen (oder Teilbeweisen) zu unterscheiden. Dies kann erreicht werden, indem die Verwendung klassischer Logik im Extrakt des Beweises sichtbar wird. Ein klassischer Kalkül, der diese Eigenschaft besitzt, ist der $\lambda\mu\nu$-Kalkül von Pym und Ritter [PR04].

Dieser Kalkül dient nun als Grundlage unserer Überlegungen. Zunächst erwägen wir, klassische Logik anhand der $\lambda\mu\nu$-Multi-Konklusion in Nuprl zu integrieren. Dieser Ansatz erweist sich zwar als ungeeignet, zeigt jedoch eine Alternative auf, die unseren Kalkül $\lambda\mu$PRL motiviert. Dieser ist ein analytischer Sequenzenkalkül, der sich stark an Nuprls Logik orientiert. Eine Implementierung in Nuprl ließe sich daher mit nur geringfügigen Änderungen des System verwirklichen.

Anschließend betrachten wir $\lambda\mu$PRL unter verschiedenen Aspekten. Wir führen eine Korrektheitsbeweis auf Basis einer gegenseitigen Simulation mit $\lambda\mu\nu$ und definieren eine konfluente Reduktions-Relation.

Zudem untersuchen wir, wie sich klassisches Schließen im Beweisterm auswirkt. Dabei zeigt sich, dass nicht jeder Term verworfen werden muss, der eines der klassischen Konstrukte enthält. Es ist vielmehr möglich, feine Unterscheidungen vorzunehmen. In diesem Zusammenhang adaptieren wir das Konzept der *μ-sicheren* Terme von Crolard [Cro02]. Zusätzlich schlagen wir eine Erweiterung dieses Konzepts vor, die noch feinere Unterscheidungen erlaubt.

Schließlich setzen wir $\lambda\mu$PRL in Bezug zu einer früheren Arbeit, die sich mit dem konstruktiven Inhalt klassischer Beweise befasst [Mur91b]. Murthy's einflussreiche Arbeit stammt ebenfalls aus dem Nuprl-Umfeld und enthält den Kalkül $PROG_K$ für Σ^0_1-Formeln der Peano Arithmetik. Wir zeigen, dass eine Variante von $\lambda\mu$PRL isomorph zu $PROG_K$ ist.

Contents

1 Introduction **1**

2 Preliminaries **4**
2.1 Classical and Intuitionistic Logic 4
2.2 The Nuprl Proof Development System 5
2.3 Murthy's $PROG_K$ 7
2.4 The $\lambda\mu$-Calculi 9
2.5 Substitution 11
2.6 Reduction 14

3 On the $\lambda\mu$-Multi-Conclusion **17**
3.1 Preliminary Considerations 17
3.2 A Multi-Conclusion for Nuprl? 20
3.3 Conclusion and Alternative Approach 22

4 $\lambda\mu$PRL **24**
4.1 The calculus $\lambda\mu$PRL 24
4.2 Consistency of $\lambda\mu$PRL 25

5 Conversion Theory **36**
5.1 Conversion in $\lambda\mu$PRL and Related Calculi 36
5.2 Confluence of $\beta\mu$-reduction 40

6 On Classical and Intuitionistic Terms of $\lambda\mu$PRL **46**
6.1 Murthy's Evaluation 48
6.2 Distinguishing Terms by Crolard's μ-safety 51
6.3 Constructive Content of $\lambda\mu$PRL-terms 58
6.4 Non-μ-safe Terms with Constructive Content 59

7 $\lambda\mu$PRL and $PROG_K$ **61**
7.1 $\lambda\mu$PRL and $PROG_K$ 61
7.2 Translation between $\lambda\mu$PRL and $PROG_K$ 62
7.3 Preservation of Reduction 66
7.4 Example Proofs 70

8 Conclusion and Perspectives **73**
8.1 Conclusion 73
8.2 Perspectives 74

List of Figures **I**

Bibliography **II**

Chapter 1

Introduction

Computational Type Theory [Con08] is the constructive type theory implemented in the Nuprl Proof development system [CAB+86, Con98]. The importance of constructive type theories for computer science results from the *proofs-as-programs* correspondence (also referred to as the Curry-Howard-isomorphism [How80]). This principle links reasoning about data types to constructive logic and thereby enables the extraction of functional programs from logical proofs. Such programs are *correct by construction* – a crucial feature in a world where sensitive domains increasingly rely on the correctness of software.

The ability to automatically synthesize a program from the proof of its specification is founded on a *constructive notion of existence*. This notion is the fundamental difference between intuitionistic (constructive) and classical logic. While in intuitionistic logic the existence of an object can only be proved if we know how to construct it, classical logic allows proofs by contradiction. As a consequence, in classical logic the existence of an object does not necessarily coincide with actually knowing the object which is supposed to exist.

Although the algorithmic knowledge how to construct an object plays an important role in computer science, modern mathematics largely depend on the classical point of view. A type theory with a classical notion of existence is e.g. implemented in the theorem prover HOL [GM93].

As a result of the different paradigms, intuitionistic logic rejects classical axioms like the law of the excluded middle or the double negation elimination. But this also makes proof search in intuitionistic logic harder: Whereas in classical propositional logic proof search is co-$\mathcal{NP}$-complete, in propositional intuitionistic logic it is $\mathcal{PSPACE}$-complete [Coo71, Sta79].

Although it was long believed that classical proofs have no constructive content, it still was a known fact that classical and intuitionistic logic can be reconciled with each other. There exist translations between intuitionistic and classical logic, e.g. Gödel's double negation translation [Gal91]. Furthermore, Friedman [Fri78] proved that classical (Peano) arithmetic is conservative over intuitionistic (Heyting) arithmetic for Π^0_2 sentences.

Then, in 1989, Griffin published a very influential paper [Gri90], proposing an extension of the proofs-as-programs-principle to classical logic. His method was to augment Church's simply typed λ-calculus [Chu40] (which is the basis for functional programming languages) with Felleisen's control operator $\mathcal{C}$ [FFKD87, FH92] and to type it with the classical axiom of double-negation-elimination. This relates classical proofs to functional programming

languages with access to the control-flow (as in imperative languages) in the same way as intuitionistic proofs are related to purely functional languages. Examples of such control constructs include the exception handling mechanism in ML or the `call/cc-` operator in Scheme.

Griffin's work triggered a lot of research concerning the semantics and constructive content of classical proofs (e.g.[BB92, BB93, RS94, Oga98, BBS95, dG95, Fuj97, Coq95, Gir91]).

Among the works inspired by Griffin, was also Murthy's research [Mur90, Mur91b, Mur91a] concerning the constructive content of Σ_1^0 sentences in Peano arithmetic. Murthy's work was situated in the context of the Nuprl system and contributed (among other results) how to directly obtain values from classical proof terms.

Another work following up Griffin's and Murthy's research was an algorithmic interpretation of natural deduction by Parigot [Par92]. For this purpose Parigot designed the classical natural deduction calculus $\lambda\mu$ which he regarded as better suited for proof-theoretic studies. Instead of the additional operator $\mathcal{C}$, the $\lambda\mu$-calculus introduced the new binding operator μ. Different from $\mathcal{C}$ the μ-operator binds a new kind of variables which (roughly speaking) denote applications of classical reasoning.

Since the $\lambda\mu$ calculus was proposed, Parigot's original version has intensively been studied (e.g. [Par92, Par93a, Py98, BHF01, Mat01]) and multiple variants of $\lambda\mu$-calculi have been defined (e.g. [OS97, Ste99, dG01, PR04]). One of these different versions is Pym's and Ritter's $\lambda\mu\nu$-calculus [PR04, RPW00, Pym, PR01].

Considering an integration of classical reasoning into computational type theory, $\lambda\mu\nu$ and the Nuprl system were the origins of this thesis. But our primary objective in this context is not the extraction of constructive content from classical proofs. We are rather interested in the creation of a hybrid system, enabling the user to decide which logic he wants to employ in different parts of a proof. Thereby the user could profit from the advantages of both concepts.

However, it is still necessary to find a way of maintaining the system's soundness in presence of both constructive and classical reasoning. That it is possible to reconcile the constructive Nuprl system's semantics with classical logic, has been shown by Howe's embedding of HOL into Nuprl [How96b, How96a]. But at the same time it is common knowledge, that systems which both validate classical axioms and allow dependent product types (which are the basis for Nuprl's constructive notion of existence) in presence of arithmetic become unsound [Mur90].

Howe solves this problem by ensuring that HOL proofs may not be used within Nuprl proofs, if a construction is required (since they have no algorithmic content but only truth values and thus cannot prove the constructive existence of an object). To implement this condition, Howe adds a parameter to proof nodes in Nuprl, which indicates whether the respective proof may be used for construction or not.

The reason to consider $\lambda\mu\nu$ for an extension of Nuprl to classical logic follows a similar line of thought: If the use of classical reasoning was visible in the proof term, this would enable us to subsequently distinguish between constructive and non-constructive proofs.

In prior work [Bre08] we already attempted a simulation of $\lambda\mu\nu$ as conservative extension of the Nuprl system. Little surprisingly, not all of $\lambda\mu\nu$'s rules were constructively provable without additional premisses: It was necessary to either supplement a non-constructive classical axiom or to state the decidability of a subformula (thereby draw-

ing a connection to the conservativity of classical over constructive logic in the decidable fragment).

However, our aim was to not restrict the range of goal formulae. Thus, it seemed a reasonable first step for this thesis to examine whether and how the $\lambda\mu\nu$-multi-conclusion could be expressed within Nuprl's type theory as real extension.

This approach helps to emphasize some aspects of the $\lambda\mu\nu$-multi-conclusion's structure and semantics. But it also turns out, that altering the conclusion part of Nuprl's sequents might not be the most sensible way to extend the system to classical logic. Instead it seems more reasonable to add a distinct second kind of (μ-)hypotheses. This view is also supported by other approaches to $\lambda\mu$ in the literature. E.g. Ong and Stewart [OS97, Ong96] use two kinds of hypotheses for λ-variables on the one hand and μ-variables on the other. It is also compatible with augmenting the hypotheses by an ,,attitude" as Stewart does in [Ste99].

Based on this, we develop the propositional calculus $\lambda\mu$PRL [1] in proof-refinement-logic-style. $\lambda\mu$PRL is shown to be equivalent to $\lambda\mu\nu$ in the sense, that the rules of each system can be simulated in the other. Moreover, it could be included into Nuprl in a convenient way.

We also define a $\beta\mu$-reduction relation for $\lambda\mu$PRL and establish its confluence.

But still, the aspect of maintaining the system's soundness remains. For this purpose we investigate the proof terms of $\lambda\mu$PRL-terms. This reveals that it is not necessary to dismiss every proof term containing one of $\lambda\mu$PRL's "classical" operators. It is rather possible to make fine distinctions between classical and constructive terms of $\lambda\mu$PRL. In this context we adapt the notion of *μ-safe* terms introduced by Crolard [Cro02] and propose an extension of this concept to *partially μ-safe* terms. This idea concerns the terms which can be evaluated to either constructive or non-constructive terms, depending on the evaluation strategy employed.

Finally, we relate $\lambda\mu$PRL to Murthy's calculus $PROG_K$ [Mur91b]. This calculus is of importance for our work as it is also situated in the environment of Nuprl and extends a part of Nuprl's logic to classical logic by an additional operator. We show that a variant of $\lambda\mu$PRL is isomorphic to $PROG_K$. This on the one hand provides a possibility to also extract computational context and on the other hand clarifies the relation between the μ- and the $\mathcal{C}$ -operator.

As to the outline of this thesis, chapter 2 contains some more background information on the systems and calculi recurring in later chapters and gives the necessary preliminary definitions. Chapter 3 contains our considerations towards a multi-conclusion-type for Nuprl and the alternative approach resulting in $\lambda\mu$PRL. In chapter 4 the calculus $\lambda\mu$PRL is developed and a mutual simulation with $\lambda\mu\nu$ is presented. Chapter 5 deals with the conversion theory of $\lambda\mu$PRL, including a confluence proof of its reduction relation. In chapter 6 we analyze how classical reasoning is visible in $\lambda\mu$PRL-terms and in chapter 7 we relate $\lambda\mu$PRLto Murthy's $PROG_K$. Finally, in chapter 8 the results of this thesis and possibilities for future work are discussed.

Acknowledgements. I would like to thank Prof. Dr. Kreitz for proposing this fascinating topic to me, giving useful advice and showing continuous interest in my work.

[1] Please note that the "μ" in $\lambda\mu$PRL is not related to Mendler's integration of μ-recursion into Nuprl's type theory [Men88]. Instead it refers to the μ-operator from the family of $\lambda\mu$-calculi.

Chapter 2

Preliminaries

This chapter is supposed to provides the preliminaries for the later chapters. Therefore we will introduce recurring calculi and systems as well as basic notation and terminology. Apart from some definitions of our own, this chapter basically contains content which can also be found in the referenced literature.

2.1 Classical and Intuitionistic Logic

Classical and intuitionistic logic share the same syntax, but differ in both their semantics and their axioms.

The crucial property of intuitionistic (also referred to as "constructive") logic is the existence property. It states that we can only claim the existence of an object if we know how to construct it. As a consequence, intuitionistic logic rejects classical tautologies like the law of the excluded middle as axiom.

Classical logic on the contrary allows proofs by contradiction, employing the law just mentioned. This means we assume that some element does not exist, and prove that this assumption is false. Then the element must exist, although we might not know how to construct - or in terms of computer science: *compute* - it.

Still, if we take an intuitionistic calculus like Gentzen's well-known $\mathcal{LJ}$ [Gen34], we can obtain a calculus for classical logic by simply adding one of the following rules:

- $\dfrac{}{A \vee \neg A}$ (*tertium non datur* / law of the excluded middle)
- $\dfrac{\Gamma, \neg\phi \vdash \bot}{\Gamma \vdash \phi}$ (*reductio ad absurdum*)
- $\dfrac{\Gamma \vdash \neg\neg\phi}{\Gamma \vdash \phi}$ (double negation elimination)

Another possibility to achieve this effect is the admission of a multi-conclusion. This approach is chosen in $\lambda\mu\nu$ (see section 2.4), while in $\lambda\mu$PRL the classical rule corresponds to reductio ad absurdum (see 4). Murthy's $PROG_K$ employs the third variant using the double negation elimination rule.

As a consequence, it is obvious that the set of intuitionistically provable formulae is a real subset of classical tautologies. Nevertheless, intuitionistic logic is not weaker than

classical logic as is shown by several translations (e.g. by Kolmogorov, Gentzen or Gödel [Gal91]). In fact, a translated formula is provable in classical logic if and only if it is provable in intuitionistic logic.

At the same time, in classical logic it is not possible to distinguish the translated formula from the original formula. This means intuitionistic logic allows to make finer distinctions. The same is valid for the different $\lambda\mu$-calculi: In these calculi the classical equivalence $P \equiv \neg\neg P$ is not true [OS97].

However, the drawback is that proof search in intuitionistic logic is harder than in classical logic: While classical propositional logic is co-$\mathcal{NP}$-complete, intuitionistic propositional logic is $\mathcal{PSPACE}$-complete [Coo71, Sta79].

2.2 The Nuprl Proof Development System

A version of Computational Type Theory is implemented in the Nuprl proof development system [CAB$^+$86, Con98] (which is now part of the *Formal Digital Library* FDL). This system does not only provide an environment for interactive proof development, but also a powerful tool for program synthesis, verification and optimization [Kre04].

Nuprl's logic is based on Martin-Löf's intuitionistic type theory [ML84]. Nevertheless it is a considerable extension of this predecessor and moreover it is open-ended. This enables the user to extend the logical language if desired.

Each type of Nuprl's type theory is associated with a set of rules. These rules concern on the one hand the type's and its members' formation and elimination, on the other hand they determine how it is evaluated and whether two instances of this type or its members are extensionally equal. This also stipulates the semantics of the type. An excerpt from Nuprl's types is given in figure 2.1, taken from [Kre04], and a more extensive overview of the current system can be found in [Kre02].

Nuprl-Expressions are defined independently of their type so that even untypable constructs like the well-known Y-combinator can be used in terms. This allows to represent all computable functions in Nuprl-terms. Still, proofs require that an expression is shown to be member of a type. Furthermore, to maintain termination of evaluation in presence of such expressiveness, Nuprl's employs a lazy evaluation strategy.

Proofs in Nuprl are carried out in a top-down sequent calculus which refines a proof goal until every subgoal is an instance of an axiom or lemma of Nuprl's library. This library contains a wide range of formalized mathematical knowledge.

The proof goal in Nuprl-proofs is not manipulated by direct application of inference rules, but by the use of proof tactics. These tactics are implemented in the system's meta-language which is a functional programming language similar to Standard ML. A tactic might in fact just carry out a single step of inference, but the concept is much more powerful. Since tactics work on the meta-level of the system, it is possible to define tactics which analyse the hypotheses and the proof goal to decide which inference rules can or should "best" (possibly depending on heuristic information) be applied.

Nuprl also enables the user to implement his own proof tactics. Furthermore, the user can even more easily combine tactics for his purposes by using *tacticals*. These allow e.g. the repeated or sequential application of predefined tactics.

Another feature of Nuprl is the possibility of defining conservative extensions of its type theory by *abstract definitions*. New types which ware defined by this mechanism

	Type	Members and associated non-canonical expressions
Function Space	$S \rightarrow T, x : S \rightarrow T$	$\lambda x.t, f\ t$
Product Space	$S \times T, x : S \times T$	$< s, t >$, `let <`x, y`>=`e `in` u
Disjoint Union	$S + T$	`inl(`s`), inr(`t`)`
		`case` e `of inl(`x`)` $\mapsto u$ `|` `inr(`y`)` $\mapsto v$
Universes	$\mathbb{U}_j$	– *types of level* j –
Equality Type	$s = t \in T$	`Ax`
Empty Type	`Void`	– *no members* – `any(`x`)`
Atoms	`Atom`	*"token"*, `if ``a''=``b'' then` $s$ `else` t
Numbers	$\mathbb{Z}$	$0, 1, -1, 2, -2, \ldots$
		`rec-case` i `of` $x < 0 \mapsto$`[`f_x`].`s`|`$0 \mapsto b$`|`$y > 0 \mapsto$`[`f_y`].`t
		$s + t, s - t, s * t, s \div t,$ `rem`t
		`if` $i = j$ `then` s `else` t, `if` $i < j$ `then` s `else` t
	$i < j$	Ax
Lists	S `list`	`[]`, t :: *list*
		`rec-case` L `of` `[]`$\mapsto b$ `|` $x :: l \mapsto$`[`f_l`].`t
Inductive Types	`rectype` $X = T[X]$	– *members defined by unrolling T[X]* –
Subset	$\{x : S \mid P[x]\}$	– *some members of* S –
Intersection	$\cap x : S.T[x]$	– *members that occur in all T[x]* –
	$x : S \cap T[x]$	– *members that occur in* S *and T[x]* –
Union	$\cup x : S.T[x]$	– *members that occur in some T[x], tricky equality* –
Quotient	$x, y : S//E[x, y]$	– *members of* S*, new equality* –
Very Dependent Functions	$\{f \mid x : S \rightarrow T[f, x]\}$	– *functions whose range types depend on the values of their inputs ans of the functions themselves* –

Figure 2.1: Types in Nuprl

Natural Numbers	$\mathbb{N}$	$\equiv$	$\{i : \mathbb{Z} \mid 0 \leq i\}$
Logical connectives	$\forall \exists \wedge \vee \Rightarrow \neg\ True\ False$		– *Curry-Howard isomorphism* –
Singleton Type	`Unit, ()`	$\equiv$	$0 = 0 \in \mathbb{Z}, Ax$
Top type	`Top`	$\equiv$	$\cap x : Void.Void$
Booleans	$\mathbb{B}$,`tt`,`ff`	$\equiv$	`Unit+Unit,inl(()),inr(())`
Boolean conditional	`if` b `then` s `else` t	$\equiv$	`case` b `of inl(_)`$\mapsto s$ `|` `inr(_)`$\mapsto t$
Y combinator	**Y**	$\equiv$	$\lambda f.\ (\lambda x.f(xx))\ (\lambda x.f(xx))$
List operations	`hd(`l`),tl(`l`),`l_1`@`l_2`,length(`l`),map(`f`;`l`),rev(`l`),`l`[`i`]`l`[`i`..`j^-`]`		

Figure 2.2: Conservative extensions in Nuprl

$$
\begin{array}{rl}
t ::== & x \mid \lambda x.t \mid (t\ t) \mid \langle t, t\rangle \mid spread(t; x, x.t) \mid \\
& \texttt{inl}(t) \mid \texttt{inr}(t) \mid decide(t; x.t; x.t) \\
& axiom \mid 0 \mid ind(t; t; x, x.t) \mid succ(t) \mid t + t \mid t \times t \\
& (\mathcal{A}\ t) \mid (\mathcal{C}\ t) \mid \lambda^v x.t \mid (t\ t)_v \mid
\end{array}
$$

Figure 2.3: $PROG_K$-terms

inherit (roughly speaking) the properties of the types used in their definition.

An example of a conservative extension is the integration of constructive logic into Nuprl via the propositions-as-types-principle (also known as the Curry-Howard-correspondence [How80]). I.e. the logical connective `and` is defined on top of the product type, `or` on the sum type, the implication on the function type and so on. Classical logic can then be used via the Gödel translation. Figure 2.2 shows some conservative extensions which are currently used in Nuprl (adopted from [Kre04]).

More information on the internal structure of the Nuprl system will be given in chapter 3 when considering the possible design of a multi-conclusion-type.

2.3 Murthy's $PROG_K$

Murthy's work [Mur91b, Mur90, Mur91a] primarily focusses on the extraction of constructive content from classical proofs, i.e. from proofs of Σ^0_1 and Π^0_2 sentences in Peano arithmetic. Since this thesis is rather aimed at the recognition of classical reasoning in a proof, our interest in Murthy's work results from its proximity to the Nuprl system.

Just like $\lambda\mu$PRL, Murthy's calculus $PROG_K$ uses a part of Nuprl's logical system (forming a calculus for the constructive Heyting arithmetic [Hey71]) and extends it to classical logic by an additional rule. In reference to Griffin [Gri90], this additional rule is the double negation elimination rule and its associated term Felleisen's control operator $\mathcal{C}$ [FFKD87, FH92].

The terms and reduction rules of $PROG_K$ are given in figure 2.3 and 2.4. Since apart from the different rules associated with the "classical" terms the rules of $PROG_K$ match the rules of $\lambda\mu_c$PRL (and thus can be found in chapter 4, figure 4.1 for the propositional and in chapter 7, figure 7.1 for the first order/ arithmetical fragment) we are content to give the double negation elimination rule and omitting the rest in this place:

$$
\begin{array}{l}
H \vdash T\ \texttt{ext}\ (\mathcal{C}\ t) \\
\quad \text{BY } \texttt{double negation elim} \\
\quad H \vdash \neg\neg T\ \texttt{ext}\ t
\end{array}
$$

Murthy on the one hand shows how to extract constructive content from classical proofs using Friedman's A-translation [Fri78]. Thereby he shows that A-translation is the proof-theoretic equivalent to a CPS-translation from imperative back to functional programs.

On the other hand, he also defines an evaluation semantics which allows him to directly extract the constructive content from a proof in Peano arithmetic. However, using the

$$
\begin{array}{rcl}
ind(0;b;x,y.u) & \rightsquigarrow_\beta & b \\
ind(k_n;b;x,y.u) & \rightsquigarrow_\beta & u[k_n,k_{n-1};b;x,y.u)/x,y] \\
decide(inl(t);x.u;y.v) & \rightsquigarrow_\beta & u[t/x] \\
decide(inr(t);x.u;y.v) & \rightsquigarrow_\beta & v[t/y] \\
spread(\langle s,t\rangle;x,y.u) & \rightsquigarrow_\beta & u[s,t/x,y] \\
((\lambda x.t)\ s) & \rightsquigarrow_\beta & t[s/x] \\
((\lambda^v x.t)\ s)_v & \rightsquigarrow_\beta & t[s/x] \text{ if } s \text{ is a value} \\
succ(k_n) & \rightsquigarrow_\beta & k_{n+1} \\
k_n + k_m & \rightsquigarrow_\beta & k_{n+m} \\
k_n \times k_m & \rightsquigarrow_\beta & k_{n\times m} \\
 & & \\
E[(\mathcal{C}\ t)] & \rightsquigarrow_c & (t\ \lambda x.(\mathcal{A}\ E[x])) \\
E[(\mathcal{A}\ t)] & \rightsquigarrow_c & t
\end{array}
$$

Figure 2.4: $PROG_K$-reduction-rules

reduction-rules which are given in figure 2.4 requires a deterministic evaluation strategy. Otherwise the intrinsic non-determinism of classical logic breaks the confluence of the calculus (this matter might become clearer in chapter 7).

In the following we present the evaluation strategy exactly as it can be found in [Mur90].

Murthy starts by defining the values of the language (which renders the strategy "lazy" as it accepts a λ-abstraction as value independently of its subexpressions) , then goes on by defining his notion of a "reducible term", which is done in terms of three different syntactic classes.

$$Val \equiv\ \lambda x.Exp_1 \mid \lambda^v x.Exp_1 \mid \langle Val_1, Val_2\rangle \mid inl(Val_1) \mid inr(Val_1)$$

- R: The class of redices which are terms that can be reduced, or – in terms of a programming language– computed
- RC: constructors which contain reducible terms as subterms
- RD: destructors the subterms of which are reducible terms

Then, the classes RC and RD are defined recursively by:

$$
\begin{array}{rl}
RC \equiv & \langle RC_1, Exp_1\rangle \\
 & \mid \langle Val_1, RC_1\rangle \\
 & \mid inl(RC_1) \mid inr(RC_1) \\
 & \mid RD
\end{array}
$$

$$
\begin{array}{rl}
RD \equiv & (RD_1\ Exp_1) \\
 & \mid (RD_1\ N)_v \\
 & \mid (\lambda^v x.Exp_1\ RD_1)_v \\
 & \mid spread(RD_1; Var_1, Var_2.Exp_2)
\end{array}
$$

$$
\begin{array}{l}
\quad \mid decide(RD_1; Var_1.Exp_2; Var_2.Exp_3) \\
\quad \mid [\,]\ the\ empty\ context)
\end{array}
$$

When we use the evaluation strategy for $\lambda\mu_c$PRL we adapt the definition as follows (the difference is of purely syntactical nature):

$$
\begin{array}{rl}
RD \equiv & (RD_1\ Exp_1) \\
& \mid (let\ \langle Var_1, Var_2\rangle = RD_1\ in\ Exp_2) \\
& \mid (case\ RD_1\ of\ inl(Var_1) : Exp_2 \mid inr(Var_2) : Exp_3) \\
& \mid [\,]\ the\ empty\ context)
\end{array}
$$

Now, Murthy gives the following notion of a reducible term based on the definitions above:

Definition 2.1. *(Reducible Term) A reducible term is one which contains a unique (possibly empty) binding-free path from the root of the term to a subterm which is a redex such that this path consists of constructors, followed by destructors, terminating in the redex.*

This definition fixes an deterministic left-to-right evaluation order on terms. The left-to-right-direction results from the parts of the definition where in presence of two subexpressions the left one is defined as RD and the right one as Exp, e.g. $\langle Val_1, RC_1\rangle$. However, this choice is arbitrary and another strategy could be chosen as long as it is deterministic.

Remark In his definition of RD Murthy leaves out a term of the form

$$ind(RD_1; Exp_2; Var_1, Var_2.Exp_3)$$

since he regards integer expressions as equivalent to their value in all contexts. The reason for this choice is to accommodate the operational semantics to the Kolmogorov-translation he uses in his work. This is possible as integer expressions may only contain integer subexpressions and hence will evaluate to a numeral at some point anyway [Mur90].

2.4 The $\lambda\mu$-Calculi

The $\lambda\mu$-calculus was first introduced by Parigot [Par92, Par93a] as an algorithmic interpretation of classical natural deduction. Although inspired by Griffin's [Gri90] and Murthy's [Mur91a] works on the constructive content of classical proofs, Parigot considered their approaches as not very well suited for proof theoretic studies. He felt that a multi-conclusioned natural deduction calculus was a better basis for his purpose.

This first version of $\lambda\mu$ is a multi-conclusioned second order calculus for the functional fragment. The μ-variables "named" the formulae of the multi-conclusion and the μ-operator marked the application of structural rules in the succedent. Parigot's $\lambda\mu$ has a confluent cut elimination procedure and is strongly normalizable [Par92, Par93a, Py98, BHF01, Mat01].

Since the $\lambda\mu$-calculus was first proposed, there has been a lot of subsequent research. Both the properties of the original calculus have been extensively studied, and numerous variants of $\lambda\mu$-calculi have been presented.

One of these variants is a $\lambda\mu$ type theory by Ong and Stewart [OS97, Ong96] which returns to a single-conclusion and keeps the non-active μ-formulae in the antecedent as μ-declarations.

Another variant is Pym's and Ritter's multi-conclusioned $\lambda\mu\nu$-calculus [PR04, RPW00, Pym, PR01] for full propositional logic (but without first or second order quantification). $\lambda\mu\nu$ was designed in the context of providing a categorial semantics for reductive logic and proof search.

In chapter 3 we start with $\lambda\mu\nu$ with regard to an integration of classical logic in to Nuprl and use it in chapter 4 for a mutual simulation with $\lambda\mu$PRL. Therefore we will in the following introduce this calculus in more detail.

The terms of $\lambda\mu\nu$ are build up from the following grammar:

$$\begin{aligned} t ::== x \mid \lambda x.t \mid (t\ t) \mid \langle t, t\rangle \mid \pi(t) \mid \pi`(t) \mid \mu\alpha.t \mid [\alpha]t \mid \\ \mu\bot.t \mid [\bot]t \mid \langle\beta\rangle\, t \mid \nu\beta.t \mid \langle\bot\rangle\, t \mid \nu\bot.t \end{aligned}$$

where x is from the alphabet of λ-variables and α from the alphabet of μ-names. These two alphabets are disjoint in $\lambda\mu\nu$.

The inference rules of $\lambda\mu\nu$ are formulated as a bottom-up sequential natural deduction calculus. As mentioned above, $\lambda\mu\nu$'s sequents are multi-conclusioned. These multi-conclusions consist of two parts: On the one hand, the active formula which provides the current type of the associated term; on the other hand, the passive "classical" context. The latter is a set of formulae which are named by μ-variables.

All rules of $\lambda\mu\nu$ except $[-]$ and $\vee^\nu E$ are constructive. As the rule μ allows to reactivate named formulae from the context or add new formulae to the conclusion, it basically corresponds to $\bot$-elimination in other calculi (although in its original formulation $\lambda\mu\nu$ treats $\bot$ separately). Symmetrical to this, applying the "classical" rule $[-]$ passivates the active type of the sequent. Thus it corresponds to the rule of $\bot$-introduction which is not valid in intuitionistic logic. (The case of the $\vee^\nu$-rules is similar.)

The possibility of exchanging the active type of the sequent reflects an intrinsic property of classical logic: Proofs may switch their proof goal along the way. To illustrate this feature, let us take a look at the proof of the law of the excluded middle - the well-known classical tautology which not intuitionistically provable.

Example 2.1. Derivation of the law of the excluded middle in $\lambda\mu\nu$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,Ax}{x : A \vdash [\alpha]x : \ \ , A^\alpha}\,[-]}{x : A \vdash \mu\bot.[\alpha]x : \bot, A^\alpha}\,\bot I}{\vdash \lambda x.\mu\bot.[\alpha]x : (A \Rightarrow \bot), A^\alpha}\,\supset I}{\vdash [\beta]\lambda x.\mu\bot.[\alpha]x : \ \ , (A \Rightarrow \bot)^\beta, A^\alpha}\,[-]}{\vdash \mu\alpha.[\beta]\lambda x.\mu\bot.[\alpha]x : A, (A \Rightarrow \bot)^\beta}\,\mu}{\vdash \nu\beta.\mu\alpha.[\beta]\lambda x.\mu\bot.[\alpha]x : A \vee (A \Rightarrow \bot)}\,\vee^\nu I$$

The treatment of the disjunction in $\lambda\mu\nu$ corresponds to locally representing the choice between the two disjuncts. This approach is similar to Gentzen's well-known classical

$$\frac{}{\Gamma x:\phi \vdash x:\phi,\Delta}\ Ax$$

$$\frac{\Gamma,x:\phi\vdash t:\psi,\Delta}{\Gamma\vdash \lambda x:\phi.t:\phi\supset\psi,\Delta}\ \supset I \qquad \frac{\Gamma\vdash t:\phi\supset\psi \qquad \Gamma\vdash s:\phi,\Delta}{\Gamma\vdash ts:\psi,\Delta}\ \supset E$$

$$\frac{\Gamma\vdash t:\phi^{\alpha},\Delta}{\Gamma\vdash \mu\alpha.t:\phi,\Delta}\ \mu \qquad \frac{\Gamma\vdash t:\phi,\Delta}{\Gamma\vdash [\alpha]\,t:\phi^{\alpha},\Delta}\ [\text{-}]$$

$$\frac{\Gamma\vdash t:\Delta}{\Gamma\vdash \mu\alpha.t:\phi,\Delta}\ \mu \qquad \frac{\Gamma\vdash t:\phi,\phi^{\alpha},\Delta}{\Gamma\vdash [\alpha]\,t:\phi^{\alpha},\Delta}\ [\text{-}]$$

$$\frac{\Gamma\vdash t:\phi,\Delta \qquad \Gamma\vdash s:\psi,\Delta}{\Gamma\vdash \langle t,s\rangle:\phi\wedge\psi,\Delta}\ \wedge I \qquad \frac{\Gamma\vdash t:\phi\wedge\psi,\Delta}{\Gamma\vdash \pi'(t):\psi,\Delta}\ \wedge E \qquad \frac{\Gamma\vdash t:\phi\wedge\psi,\Delta}{\Gamma\vdash \pi(t):\phi,\Delta}\ \wedge E$$

$$\frac{\Gamma\vdash t:\Delta}{\Gamma\vdash \mu\bot.t:\bot,\Delta}\ \bot I \qquad \frac{\Gamma\vdash t:\bot,\Delta}{\Gamma\vdash [\bot]\,t:\Delta}\ \bot E$$

$$\frac{\Gamma\vdash t:\phi,\Delta}{\Gamma\vdash \nu\bot.t:\phi\vee\bot,\Delta}\ \vee^{\nu} I_{\bot} \qquad \frac{\Gamma\vdash t:\phi\vee\bot,\Delta}{\Gamma\vdash \langle\bot\rangle\,t:\phi,\Delta}\ \vee^{\nu} E_{\bot}$$

$$\frac{\Gamma\vdash t:\phi,\psi^{\beta},\Delta}{\Gamma\vdash \nu\beta.t:\phi\vee\psi,\Delta}\ \vee^{\nu} I \qquad \frac{\Gamma\vdash t:\phi\vee\psi,\Delta}{\Gamma\vdash \langle\beta\rangle\,t:\phi,\psi^{\beta},\Delta}\ \vee^{\nu} E$$

Figure 2.5: The inference rules of $\lambda\mu\nu$

sequent calculus $\mathcal{LK}$ [Gen34]. However, Pym and Ritter mention the possibility of using a constructive disjunction instead. This would result in a calculus $\lambda\mu\oplus$ which is equivalent up to mutual simulation. We exploit this fact in chapter 4.

Another alternative formulation of $\lambda\mu\nu$ or $\lambda\mu\oplus$ concerns the handling of $\bot$. Alternatively to including additional rules and terms for $\bot$, these could be combined with the rules μ ans $[-]$. In this case the active type of the sequent is not empty, when a formula has been moved to the context, but instead the active type becomes $\bot$. We denote the respective variants as $\lambda\mu\nu_{\bot}$ and $\lambda\mu\oplus_{\bot}$, and the latter is the basis for our considerations in chapter 3.

2.5 Substitution

We require two different notions of substitution. On the one hand the usual substitution of the λ-calculus, on the other hand structural substitution as common for $\lambda\mu$-terms. While the former replaces free occurrences of λ-variables and avoid variable-capture, the latter is a textual replacement within a term and we need the notion of a *context*, i.e. a term $C[-]$ with a hole.

Since we deal with three different calculi, we have to define both kinds of substitution for all of them. However structural substitution for $PROG_K$ is basically an abbreviation to simplify the presentation of the proofs in chapter 7.

Definition 2.2. *(Substitution of λ-variables in $\lambda\mu PRL$) The substitution $t[s/x]$ where x is a λ-variable and s an arbitrary $\lambda\mu PRL$-term is defined inductively by:*

- $y^* = y \quad if\ y \neq x$
- $x^* = s$
- $(\lambda y.t)^* = \lambda y.(t^*) \quad if\ y \neq x$
- $(\lambda x.t)^* = \lambda x.t$
- $(s\ t)^* = (s^*\ t^*)$
- $\langle s, t\rangle^* = \langle s^*, t^*\rangle$
- $(let\ \langle y_1, y_2\rangle = z\ in\ t)^* = let\ \langle y_1, y_2\rangle = (z^*)\ in\ (t^*)$
  $if\ y_1 \neq x\ and\ y_2 \neq x$
- $(let\ \langle x, y\rangle = z\ in\ t)^* = let\ \langle x, y\rangle = (z^*)\ in\ t$
  $if\ y_1 = x\ or\ y_2 = x$
- $(inl(t))^* = inl(t^*)$
- $(inr(t))^* = inr(t^*)$
- $(case\ z\ of\ inl(y_1) : u \mid inr(y_2) : v = case\ (z^*) of\ inl(y_1) : (u^*) \mid inr(y_2) : (v^*)$
  $if\ y_1 \neq x\ and\ y_2 \neq x$
- $(case\ z\ of\ inl(y_1) : u \mid inr(y_2) : v = case\ (z^*) of\ inl(y_1) : u \mid inr(y_2) : (v^*)$
  $if\ y_1 = x$
- $(case\ z\ of\ inl(y_1) : u \mid inr(y_2) : v = case\ (z^*) of\ inl(y_1) : (u^*) \mid inr(y_2) : v$
  $if\ y_2 = x$
- $(\mu\alpha.t)^* = \mu\alpha.(t^*)$
- $([\alpha]t) = [\alpha](t^*)$

where t^* stands for $t[s/x]$. If x occurs freely in s it must be substituted by a fresh variable x' before the substitution: $s[x'/x]$.

Definition 2.3. *(Substitution in $\lambda\mu_c PRL$)*
The substitution $t[s/x]$ in $\lambda\mu_c PRL$ extends the definition of substitution in $\lambda\mu PRL$. The inductive definition for all common terms is the same as in $\lambda\mu PRL$. The inductive definition to the additional terms is as follows:

- $(\lambda^v y.t)^* = \lambda^v y.(t^*) \quad if y \neq x$
- $(\lambda^v x.t)^* = \lambda^v x.t$
- $(s\ t)^*_v = (s^*\ t^*)_v$
- $(\mathcal{A}\ t)^* = (\mathcal{A}\ (t^*))$
- $axiom^* = axiom$
- $0^* = 0$
- $ind(s; t; y_1, y_2.v)^* = ind((s^*); (t^*); y_1, y_2.(v^*)) \quad if\ y_1 \neq x\ and\ y_2 \neq x$
- $ind(s; t; y_1, y_2.v)^* = ind((s^*); (t^*); y_1, y_2.v) \quad if\ y_1 = x\ or\ y_2 = x$
- $succ(t)^* = succ(t^*)$
- $(s + t)^* = (s^*) + (t^*)$
- $(s \times t)^* = (s^*) \times (t^*)$

where t^* stands for $t[s/x]$. If x occurs freely in s it must be substituted by a fresh variable x' before the substitution: $s[x'/x]$.

Definition 2.4. *(Substitution in $PROG_K$)*
The substitution $t[s/x]$ where t and s are terms of $PROG_K$ and x is a λ-variable is inductively defined by :

- $y^* = y \quad if\ y \neq x$
- $x^* = s$
- $(\lambda y.t)^* = \lambda y.(t^*) \quad if\ y \neq x$
- $(\lambda x.t)^* = \lambda x.t$
- $(s\ t)^* = (s^*\ t^*)$
- $\langle s, t\rangle^* = \langle s^*, t^*\rangle$
- $spread(z; x, y.t)^* = spread((z^*); y_1, y_2.(t^*) \quad if\ y_1 \neq x\ and\ y_2 \neq x$
- $spread(z; x, y.t)^* = spread((z^*); y_1, y_2.t \quad if\ y_1 = x\ or\ y_2 = x$
- $(inl(t))^* = inl(t^*)$
- $(inr(t))^* = inr(t^*)$
- $decide(z; y_1.u; y_2.v)^* = decide((z^*); y_1.(u^*); y_2.(v^*)) \quad if\ y_1 \neq x\ and\ y_2 \neq x$
- $decide(z; y_1.u; y_2.v)^* = decide((z^*); y_1.(u^*); y_2.v) \quad if\ y_2 = x$
- $decide(z; y_1.u; y_2.v)^* = decide((z^*); y_1.u; y_2.(v^*)) \quad if\ y_1 = x$
- $(\mu\alpha.t)^* = \mu\alpha.(t^*)$
- $([\alpha]t) = [\alpha](t^*)$
- $(\lambda^v y.t)^* = \lambda^v y.(t^*) \quad if\ y \neq x$
- $(\lambda^v x.t)^* = \lambda^v x.t$
- $(s\ t)^*_v = (s^*\ t^*)_v$
- $(\mathcal{A}\ t)^* = (\mathcal{A}\ (t^*))$
- $axiom^* = axiom$
- $0^* = 0$
- $ind(s; t; y_1, y_2.v)^* = ind((s^*); (t^*); y_1, y_2.(v^*)) \quad if\ y_1 \neq x\ and\ y_2 \neq x$
- $ind(s; t; y_1, y_2.v)^* = ind((s^*); (t^*); y_1, y_2.v) \quad if\ y_1 = x\ or\ y_2 = x$
- $succ(t)^* = succ(t^*)$
- $(s + t)^* = (s^*) + (t^*)$
- $(s \times t)^* = (s^*) \times (t^*)$

where t^* stands for $t[s/x]$.If x occurs freely in s it must be substituted by a fresh variable x' before the substitution: $s[x'/x]$.

Definition 2.5. *(Structural Substitution in $\lambda\mu PRL$)*
The structural substitution $t[[\alpha]C[u]/[\alpha]u]$ is defined inductively as follows:

- $x^* = x$
- $(\lambda x.t)^* = \lambda x.(t^*)$
- $(s\ t)^* = (s^*\ t^*)$
- $\langle s, t\rangle^* = \langle s^*, t^*\rangle$
- $(let\ \langle x, y\rangle = z\ in\ t)^* = let\ \langle x, y\rangle = (z^*)\ in\ (t^*)$
- $(inl(t))^* = inl(t^*)$
- $(inr(t))^* = inr(t^*)$

- $(case\ z\ of\ inl(x) : u \mid inr(y) : v = case\ (z^*) of\ inl(x) : (u^*) \mid inr(y) : (v^*)$
- $(\mu\beta.t)^* = \mu\beta.(t^*) \quad if\ \alpha \neq \beta$
- $(\mu\alpha.t)^* = \mu\alpha.t$
- $([\beta]t) = [\beta](t^*)$
- $([\alpha]t)^* = [\alpha]C[(t^*)]$

where t^* stands for $t[[\alpha]C[u]/[\alpha]u]$.

Definition 2.6. *(Structural Substitution in $\lambda\mu_c PRL$)*
The structural substitution $t[(\mathcal{A}\ C[u])/[\alpha]u]$ in $\lambda\mu_c PRL$ is an for the most part an extension of the structural substitution in $\lambda\mu PRL$. The inductive definition for all terms except $s \equiv [\alpha]t$ is the same as in $\lambda\mu PRL$. The altered definition for s and the extension of the inductive definition to the additional terms is as follows:

- $([\alpha]t)^* = (\lambda x.(\mathcal{A}\ C[x])\ (t^*))$
- $(\lambda^v x.t)^* = \lambda^v x.(t^*)$
- $(s\ t)^*_v = (s^*\ t^*)_v$
- $(\mathcal{A}\ t)^* = (\mathcal{A}\ (t^*))$
- $axiom^* = axiom$
- $0^* = 0$
- $ind(s;t;x,y.v)^* = ind(s;t;x,y.v)$
- $succ(t)^* = succ(t^*)$
- $(s+t)^* = (s^*) + (t^*)$
- $(s \times t)^* = (s^*) \times (t^*)$

where t^* stands for $t[(\mathcal{A}\ C[u])/[\alpha]u]$.

Definition 2.7. *(Structural substitution for $PROG_K$-terms)*
Structural substitution for $PROG_K$ is defined on top of the regular substitution. For all $PROG_K$-terms t:

$$t[(\mathcal{A}\ C[t])/(\alpha\ t)] = t[\lambda x.(\mathcal{A}\ C[x])/\alpha]$$

2.6 Reduction

In chapter 5 we need some background knowledge concerning reduction in term-calculi. The necessary notions are introduced in the following. The definitions are standard and adopted from Barendregt [Bar84] and Py [Py98].

Definition 2.8. *(Notion of Reduction)*

1. *A notion of reduction on a set of terms Σ is a binary relation R on Σ.*
2. *If R_1, R_2 are notions of reduction, then $R_1 R_2$ is $R_1 \cup_2$.*

Definition 2.9. *(Reduction Relation)*

1. *A binary relation R on a set of terms Σ is compatible (with the operations) if $(M, M') \in R$ implies that $(C[M], C[M']) \in R$ where $C[-]$ is a context with one hole and $M, M' \in \Sigma$.*

2. *An equality (or congruence) relation on Σ is a compatible equivalence relation.*

3. *A reduction relation on Σ is one which is compatible, reflexive, and transitive.*

Definition 2.10. *If $\rightarrow$ is a binary relation R on a set X, then the reflexive closure is the least relation extending R that is reflexive. The transitive and compatible closure are defined similarly.*

Definition 2.11. *($\rightarrow^*, \rightarrow^+, \rightarrow^=, \rightarrow^n$)*
Let R be a notion of reduction on Σ. Then R induces the binary relations:

$\rightarrow_R$ *one step R-reduction*
$\twoheadrightarrow_R$ *R-reduction*
$\rightarrow^+$ *transitive closure*
$=_R$ *R-equality*

where

- $\rightarrow_R$ *is the compatible closure of R*
- $\twoheadrightarrow_R$ *is the transitive, reflexive closure of R*
- $=_R$ *is the equivalence relation generated by $\twoheadrightarrow_R$:*

  1. *$M \twoheadrightarrow_R N$ implies $M =_R N$*
  2. *$M =_R N$ implies $N =_R M$*
  3. *$M =_R N$,$N =_R L$ implies $M =_R L$*

Lemma 2.12. *The relations $\twoheadrightarrow_R$, $\rightarrow^+$ and $=_R$ are all compatible. Therefore $\twoheadrightarrow_R$ is a reduction relation and $=_R$ is an equality relation.*

Proof. The proof by induction can be found in Barendregt's book [Bar84], p.52. □

Definition 2.13. *(Abstract Reduction System)*
An abstract reduction system is a pair $\langle \Sigma, (\rightarrow_i)_{i \in I} \rangle$ where Σ is a set of terms and $(\rightarrow_i)_{i \in I}$ a set of binary relations (called reductions) on Σ such that for every element M of Σ the set $\{M' | (\exists i \in I)(M \rightarrow_i M')\}$ is finite.

Example 2.2. $\lambda\mu$PRL is an abstract reduction system: Σ is the set of $\lambda\mu$PRL-terms and the reductions are the β- and μ-reduction as defined in chapter 5.

The following notions are important when considering the proof-theoretic properties of a reduction system.

Definition 2.14. *(Local Confluence) An abstract reduction system $\langle \Sigma, \rightarrow \rangle$ is said to be confluent if for all $M, M'.M'' \in \Sigma$ there is an $N \in \Sigma$ such that: if [$M \rightarrow M'$ and $M \rightarrow M''$] then [$M' \twoheadrightarrow N$ and $M'' \twoheadrightarrow N$].*

Definition 2.15. *(Confluence) An abstract reduction system $\langle \Sigma, \rightarrow \rangle$ is said to be confluent if for all $M, M'.M'' \in \Sigma$ there is an $N \in \Sigma$ such that: if $[M \twoheadrightarrow M'$ and $M \twoheadrightarrow M'']$ then $[M' \twoheadrightarrow N$ and $M'' \twoheadrightarrow N]$.*

Definition 2.16. *(Strong Confluence) An abstract reduction system $\langle \Sigma, \rightarrow \rangle$ is said to be confluent if for all $M, M'.M'' \in \Sigma$ there is an $N \in \Sigma$ such that: if $[M \rightarrow M'$ and $M \rightarrow M'']$ then $[M' \rightarrow N$ and $M'' \rightarrow N]$.*

Definition 2.17. *(Normal Form)*
In an abstract reduction system $\langle \Sigma, (\rightarrow_i)_{i \in I} \rangle$ a term M is normal if the set $\{M' | (\exists i \in I)(M \rightarrow_i M')\}$ is empty.

Definition 2.18. *(Strong Normalization) In an abstract reduction system a term M is strongly normalizable if there is no infinite sequence of reductions starting in M.*
An abstract reduction system is strongly normalizing if all of its terms are strongly normalizable.

Definition 2.19. *(Weak Normalization) In an abstract reduction system a term M is weakly normalizable if there is at least one finite sequence of reductions starting in M.*
An abstract reduction system is weakly normalizing if all of its terms are weakly normalizable.

Definition 2.20. *(Uniqueness of Normal Form) In a confluent abstract reduction system the normal form of a term is unique.*

Definition 2.21. *(Termination) An abstract reduction system terminates, if it is strongly normalizing.*

Lemma 2.22. *A strongly confluent abstract reduction system is confluent.*

For further information on reduction in the λ-calculus or term rewriting systems in general we refer the reader to Barendregt [Bar84] and Klop [Klo92].

Chapter 3

On the $\lambda\mu$-Multi-Conclusion

In this chapter we discuss different approaches to an integration of the $\lambda\mu$-multi-conclusion into the Nuprl system.

As a basis for our considerations we use the $\lambda\mu\oplus_\perp$-calculus, a variant of Pym's and Ritter's $\lambda\mu\nu$-calculus. The rules of this version can be found in figure 4.4 and a proof of its equivalence to $\lambda\mu\nu$ is given in chapter 4. We will briefly address our motivation why to use this variant in the first section of this chapter. We will also describe the results of our attempts at defining a multi-conclusion as conservative extension of Nuprl's type theory.

Then we approach a real extension of the system. There are two basic ways to include a multi-conclusion into Nuprl: Either by defining a new type of multi-conclusions or by changing the system's internal notion of a sequent. We will pursue both ideas in section 3.2.

Finally, in section 3.3 we will draw some conclusions from our considerations and propose an alternative approach. This will result in the calculus $\lambda\mu$PRL which is presented in chapter 4.

3.1 Preliminary Considerations

The obvious initial step towards a $\lambda\mu$-style multi-conclusion for Nuprl is to choose which of the available $\lambda\mu$-calculi is best suited. However, to motivate this decision, it is necessary to give a general idea of the $\lambda\mu$-multi-conclusion's structure at first.

Its structure particularly differs from the "usual" multi-conclusion of classical sequent calculi such as Gentzen's $\mathcal{LK}$ [Gen34]. This traditional notion of a multi-conclusion basically treats its formulae as "equals". Therefore the comma delimiter is semantically equivalent to the classical `or`. Contrary to this, the $\lambda\mu$-multi-conclusion consists of two distinct main parts:

The active formula determines the type of the current term. It is also the formula on which the inference rules operate.

The passive context is separated from the active formula by a comma. This comma however is not semantically equivalent to the classical `or`. The context is a set of an arbitrary number of ,,named" formulae. These names are labels which can be interpreted as a distinct kind of variables: the μ-variables. The μ-variables encapsulate the formulae associated with them such that they can only be accessed by the

specific inference rules μ and $[-]$. These two rules perform the exchange operations between context and active position.

This asymmetrical structure is the trick that allows to control the application of classical reasoning in presence of a multi-conclusion. At the same time it means, that to some extent, the context formulae are independent of the conclusion. After all they can exclusively be accessed by the exchange rules, not by any of the logical rules.

Being aware of the $\lambda\mu$-multi-conclusion's structure, we can now motivate the choice of $\lambda\mu\oplus_\perp$ as a basis.

In $\lambda\mu\nu$ (like in some other versions of $\lambda\mu$) the active position of the conclusion is empty during the exchange operation, i.e. the current proof-term has no type. So the question is how to implement this concept. One could certainly define a special "non"-type. But if we take a closer look at Nuprl types, there already is an empty type: `void`, the data type which is also used for the definition of logical absurdity $\perp$. Corresponding to this, $\perp$ is exactly the type of the term during an exchange in $\lambda\mu\oplus_\perp$. Thus, preferring $\lambda\mu\oplus_\perp$ to $\lambda\mu\nu$ is the straightforward solution.

Now, if this were the only aspect to consider, we could also use $\lambda\mu\nu_\perp$ instead of $\lambda\mu\oplus_\perp$. But the reason to choose $\lambda\mu\oplus_\perp$ over $\lambda\mu\nu_\perp$ is equally straightforward. The difference between these two variants is the treatment of disjunction. For their purpose, Pym and Ritter prefer a local representation of the choice between the two disjuncts. Therefore in $\lambda\mu\nu$ and $\lambda\mu\nu_\perp$ the right-hand disjunct is directly moved to the context. In $\lambda\mu\oplus_\perp$ on the contrary, the treatment of the disjunction resembles intuitionistic logic. Accordingly, the corresponding proof terms are the same as in Nuprl. As a consequence $\lambda\mu\oplus_\perp$ is preferable since it does not require the additional terms like $\lambda\mu\nu_\perp$ would.

Remark Still, the second disjunct is not necessarily "lost" as in the intuitionistic case. Backtracking is possible if the disjunction has been saved in the passive context beforehand. This is important for the equivalence of $\lambda\mu\nu_\perp$and $\lambda\mu\oplus_\perp$.

Before we start our reflection on the actual ways to extend Nuprl on the basis of $\lambda\mu\oplus_\perp$, let us summarise the results of [Bre08]. In this work , the author considered and implemented a simulation of $\lambda\mu\nu_\perp$and $\lambda\mu\oplus_\perp$in Nuprl. For this purpose we defined the multi-conclusion and its elements as conservative extension of Nuprl's type theory. Conservative extensions in Nuprl can be implemented via abstract definitions on top of existing types. Our definition of the multi-conclusion resulted in three new constructs: the `mu-name`, the `context` and the `komma`. The latter referred to the connection between active and passive part of the multi-conclusion, hence its name.

As we were interested in the semantics of the comma delimiter of the multi-conclusion, we examined three different variants (in the following, $\vee$ is the intuitionistic `or`) :

1. `komma1`$(A; P) == A \vee P$
2. `komma2`$(A; P) == \neg(\neg A \wedge \neg P)$
3. `komma3`$(A; P) == \neg P \Rightarrow A$

where A is the placeholder for the active and P for the passive part of the conclusion. The motivation to use these formulations were the different notions of disjunction they

represent. While the `komma1` is defined by the constructive version, `komma2` and `komma3` exploit the classical equivalences of disjunction. Obviously each of these definitions triggers different definitions for the passive context and its elements:

1. $\texttt{context1}(P1; P) == P1 \vee P$
   $\texttt{mu-name1}(\alpha 1; P1) == \alpha 1 \in \texttt{Atom} \wedge P1$
2. $\texttt{context2}(P1; P) == \neg(\neg P1 \wedge \neg P)$
   $\texttt{mu-name2}(\alpha 1; P1) == \alpha 1 \in \texttt{Atom} \wedge P1$
3. $\texttt{context3}(P1; P) == P1 \wedge P$
   $\texttt{mu-name3}(\alpha 1; P1) == \alpha 1 \in \texttt{Atom} \wedge \neg P1$

Example 3.1. Now a multi-conclusion $A, P1^{\alpha}$ of $\lambda\mu\oplus_{\perp}$ is be simulated by

$$\texttt{komma}_\texttt{i}(A; \texttt{context}_\texttt{i}(\texttt{mu-name}_\texttt{i}(\alpha 1; P1); \texttt{ic}_\texttt{i})))$$

where `ic` stands for "initial context". This construct is defined by the neutral element of the respective junctor. E.g. $\perp$ for `context1`, being the neutral element of disjunction.

Having defined all necessary constructs, the rules of the two calculi were formulated as lemma and proven within the Nuprl system. Once proven, these lemma can be used to simulate $\lambda\mu\nu_{\perp}$- of $\lambda\mu\oplus_{\perp}$-proofs in Nuprl.

Example 3.2. The rule μ of $\lambda\mu\oplus_{\perp}/\lambda\mu\nu_{\perp}$formulated as a lemma:

$$\forall A, D : \mathbb{P}.\forall\alpha : \texttt{Atom}.\texttt{komma}_i(\perp;\texttt{context}_i(\texttt{mu-name}_i(\alpha; A); D)) \Rightarrow \texttt{komma}_i(A; D)$$

These rule simulations illustrated very clearly the semantics of $\lambda\mu$'s multi-conclusion. Not surprisingly, none of the definitions allowed to prove all rule-lemmas in Nuprl's constructive logic. At least one rule always required an additional premiss, which stated the decidability of a subformula. The most adequate simulation obviously was `komma3`. Using `komma3` in the lemmas located the non-constructivity of the calculi in the exchange rules (right where it belongs). Contrarily, simulating the rules with `komma1` and `komma2` resulted in a non-constructive implication introduction.

The outcome of these simulations can be seen as illustration of the roles of the exchange-rules in $\lambda\mu\oplus_{\perp}$. Consider the following inference steps:

$$\dfrac{\dfrac{\dots}{\Gamma \vdash [\alpha]t : C, \Delta}\,[-]}{\Gamma \vdash t : \perp, C^{\alpha}, \Delta}\,\dots \qquad \dots \qquad \dfrac{\dfrac{\dots}{\Gamma \vdash \mu.t : \perp, C^{\alpha}, \Delta}\,\mu}{\Gamma \vdash t : C, \Delta}\,\dots$$

It is quite obvious that $[-]$ takes the place of an introduction rule for $\perp$. This is one of the non-constructive rules which can be used to obtain a classical calculus from an intuitionistic one (see chapter 2.1). And this is appropriately reflected, if `komma3` is used for the simulations.

As a consequence, we could interpret a sequent

$$H \vdash A, P$$

of $\lambda\mu\oplus_\perp$ as, e.g. sequent of $\mathcal{LK}$ by

$$H, P^\neg \vdash A$$

where $P^\neg$ arises from P by replacing every $P_i^{\alpha_i}$ by $\neg P$.

This interpretation indicates that the a multi-conclusion might not be the best approach in extending Nuprl. Instead, an elegant way could be to treat the passive formulae of the conclusion as a distinct kind of hypotheses. From a syntactical point of view, the results of the simulation also mean that an extension of Nuprl's inference logic would merely require the two exchange rules and their associated terms.

In the next section we nevertheless follow two different lines of thought concerning an explicit extension of Nuprl by a $\lambda\mu$-multi-conclusion. These considerations will substantiate the observations made above and lead us to the conclusions drawn in section 3.3.

3.2 A Multi-Conclusion for Nuprl?

As mentioned above, there are two basic approaches to realizing a $\lambda\mu\oplus_\perp$-multi-conclusion in Nuprl:

1. By defining a new multi-conclusion type or
2. By changing the internal representation of Nuprl's sequents

We will start by considering the first approach. Explicitly, this means to regard $\lambda\mu$'s multi-conclusion as semantic concept, not as syntactic element of the sequent.

This proceeding has two natural advantages: We would not have to alter the underlying structure of the system and whenever we wanted to use classical logic, we would need this type to encapsulate the goal formula. This could easily be filtered, to keep the classical and the constructive system apart. However, this would also limit the user's freedom, requiring to choose beforehand whether classical logic is to be employed or not.

Now, when planning a new type, we have to consider the following aspects:

- the structure of the type,
- its canonical members and the non-canonical terms associated with this type
- the reduction of non-canonical forms associated with the type
- extensional equality: under which conditions can two instances of the type or its members be considered as extensionally equal
- the inference rules associated with the type: Type formation, membership, equality and elimination; these rules need to capture the definitions from above

Reasonably, we should start by analyzing the structure of the new multi-conclusion type, denoted by `mc` in the following. Primitive types in Nuprl are implemented as abstract types of the underlying metalanguage ML. These internal definitions determine the operator-name, the number of subterms and variable-bindings.

The $\lambda\mu$-multi-conclusion consists of two parts, the active formula and the passive context (as described in the previous section). Accordingly, `mc` also requires two parts. The active part may without question be an arbitrary term. However, the part representing the context could be treated in different ways.

We could define another type `mucontext`, having formulae of another type `muname` as subterms. But: The μ-names are another kind of variables, bound by the new binding operator μ. Furthermore we have seen in the previous section, that the context-formulae can be interpreted as negated hypotheses. Thus μ-variables are to some extent symmetrical to λ-variables and this should be reflected in the data structure.

The antecedent of a Nuprl sequent is defined as `declaration list`; a `declaration` itself is a pair of a variable and a term. Therefore it might be appropriate use as list of type `mudec` (short for mu-declaration) as passive context. Then `mudec` should be a pair of a variable and a term, corresponding to `declaration`.

Instead of defining an additional construct, we could also simply use a list of products of variables and terms. However, most importantly, there would have to be a mechanism of the meta-system allowing to access the formula of the context list.

As next step, we have to consider the elements of this type. Since `mc` is supposed to represent the essence of classical reasoning, one might initially expect, that the members of `mc` needed to be the terms $\mu\alpha.t$ and $[\alpha]t$. But if we take a closer look and remember that the multi-conclusion can only be used if it encapsulates the goal-formula – then we realize that this means all applicable rules have to be associated with this type, Nuprl's existing rules cannot work on the active type. (That the use of any existing rules would lead to odd results is ignored for now, as it is merely technical.)

Now, also the terms associated with the respective active types of `mc` would need to be associated with the new type. Otherwise any type to be provable "within" `mc` would require another multi-conclusion-rule.

Example 3.3. Assume the sequent is $H \vdash A \wedge B, D^{\alpha}$. In $\lambda\mu\oplus_{\perp}$ the active position determines the type of the current term, so the current term has to be a pair $\langle a, b\rangle$. But we can not apply the usual introduction rule for $\wedge$, since it expects a conclusion of the form $H \vdash A \wedge B$. Thus we need an additional rule to handle the multi-conclusion. This new rule could either be associated with $\wedge$ (which is the product type in Nuprl) or with `mc`.

Keeping in mind, that the semantics of a Nuprl sequent is that the extract term is proved to be a member of the type in the conclusion, none of the two solutions seems appropriate. In the usual Nuprl semantics, `pair` would now be a canonical member of `mc`, too. But associating an `mc`-rule with the product type would contradict the aforementioned principle of type-membership.

After these observations, it is already obvious, that the definition of a multi-conclusion type might not be a neat extension of Nuprl's type theory. Still, it could be used to explicitly simulate a multi-conclusioned calculus in Nuprl, although this is not what we are interested in.

Now it seems futile to finally consider under which premises two instances of `mc` could be regarded as extensionally equal. But let us follow through with a naive approach. On first thought, two multi-conclusions could either be equal, if all of their corresponding elements are equal, or, if the type in their active positions is equal. In the latter case,

the associated proof terms have the same type. But on second thought, both notions of equality are questionable: If we simply compare the active types, we ignore that the type might be switched to one of the passive types; if we compare all elements we might also compare passive formulae which are not even used in the proof.

All of these problems are not only caused by the inopportune approach to define a multi-conclusion type, but also by the inherent non-determinism of classical logic (a recurring issue in chapters 5 to 7).

We have already seen, that the multi-conclusion might not be a good way to extend Nuprl. Let us still take a look at the second approach to include a multi-conclusion into Nuprl.

The second approach is of syntactic nature and requires to alter Nuprl's internal notion of a sequent. Then the semantic of Nuprl's sequents would have to be adjusted to make sure the active type was the crucial one for all membership and equality issues. Otherwise, the existing theories would become nonsensical.

The use of the μ-exchange rules (and thus classical reasoning) would have to be revealed by the proof term. This coincides with our original anticipation.

The elements of the multi-conclusion could be internally represented in the same way as we imagined for the type `mc` before. This means the μ-variables would be implemented analogously to the λ-variable. According to this symmetry, the terms $\mu\alpha.t$ and $[\alpha]$ should also be oriented on the terms of λ-abstraction and application and implemented in the same way. The new rules could be considered as structural rules, not associated with a certain type.

This approach seems to more reasonable than the first one, but it has one major drawback: Since it profoundly changes the internal structure of the system, it would require to adapt all of Nuprl's numerous existing theories to the new representation. This completely contradicts our paradigm to minimize the changes necessary in the existing system.

However, all of our consideration have a common drift towards an elegant solution which will be presented in the next section.

3.3 Conclusion and Alternative Approach

Although the idea of integrating a $\lambda\mu$-multi-conclusion into Nuprl did not turn out very well, the naive approach helped to emphasize some important aspects:

- the μ-variables correspond to the λ-variables: the passive context formulae can be regarded as negated hypotheses
- the type of the proof term is always determined by the active part of the sequent like in the single-conclusioned Nuprl system
- the syntactic position of the passive part is irrelevant as it can be accessed exclusively by the respective exchange rules; this exclusive access also allows to identify switches and thus classical reasoning
- the possibility of switching the active proof goal complicates equality reasoning: occurrences of μ-constructs in proof-terms have to be analyzed to maintain the

soundness of the system (the semantic of proof term being constructive evidence for the type of the conclusion becomes questionable if the active type can be exchanged)

- the exchange rules correspond to the rules for logical absurdity and thus also to the empty data type `void`

The first three observations provide the basis for developing $\lambda\mu$PRL in chapter 4 and will be reflected in the calculus' structure. As to the relation between λ- and μ-variables, it has to be mentioned that it results from the origins of $\lambda\mu$. This will become obvious in chapter 7, when we relate $\lambda\mu$PRL to Murthy's $PROG_K$ [Mur91b].

The difficulties that became apparent when considering equality in presence of classical logic are omnipresent in the context of reduction and evaluation of classical term calculi. They are related to the inherent non-determinism of classical logic and reflected in the fact that reduction in $\lambda\mu$ calculi is not always confluent. This problem will be discussed concerning reduction in chapter 5 and deterministic evaluation strategies in chapters 6 and 7.

The observation that the exchange rules of $\lambda\mu$-calculi correspond to logical absurdity draws a connection between classical logic an untypable λ-terms (e.g. the well-known `Y`-combinator). It can be shown, that in Nuprl such terms would become typable, if `void` was not empty (i.e. the logical absurdity was inhabited). Now, according to Nuprl's semantics the rule `abort` of $\lambda\mu$PRL (see figure 4.1) means that some term $[\alpha]t$ *is* a member of `void`. The reason for this apparent "contradiction" is the non-constructive notion of existence in classical logic. But a solution to this problem lies in the special structural reduction of $\lambda\mu$-calculi. As the *mu*-operator can be regarded as a jump operator, it can e.g. "jump" out of a while-loop. Although at the moment we do not pursue this relation any further, it is an interesting topic for future work (see chapter 8).

Concluding this chapter, we give some ideas towards an implementation which treats the passive formulae of the $\lambda\mu$-multi-conclusion as special kind of hypotheses. Roughly, the following steps would suffice for an experimental implementation:

1. Internally add the concept of μ-variables and "μ-hypotheses" and extend the definition of a sequent to handle μ-hypotheses (this would be less costly than the changing the conclusion part of the sequent, since changing the declaration list to a list of type disjoint union of two kinds of declarations does not interfere with existing concepts).

2. Define the new operators μ and $[-]$ as abstract data types of ML.

3. Implement the "classical" rules on top of the new structures.

4. Mark proofs containing one of the new constructs in their proof term by some label to ensure the soundness of the system.

Chapter 4

$\lambda\mu$PRL

Based on the considerations presented in the last chapter, we now formulate the analytic, single-conclusioned sequent calculus $\lambda\mu$PRL. In the first section of this chapter we will give both a general idea and the precise formulation of the calculus. The second section contains a proof of the consistency of $\lambda\mu$PRL.

4.1 The calculus $\lambda\mu$PRL

The calculus is based on the logical rules of the Nuprl system, although in Nuprl they are associated with the types on which the logical connectives are defined via the Curry-Howard-isomorphism [How80].

In $\lambda\mu$PRL, the passive formulae of the μ-context are positioned on the left-hand side of the sequent (similar to Ong and Stewart[Ong96, OS97]), and distinguished from the λ-context by its syntactic form. However its role is equivalent to the context part of Pym's and Ritter's $\lambda\mu\nu$-multi-conclusion since it is separated from the rest of the sequent by its names (i.e. the μ-variables) and accessible only by the specific exchange rules. This also coincides with Stewart's [Ste99] formulation of a $\lambda\mu$-theory which augments every hypothesis with an *attitude*, interpreting the μ-variables as *refutations*. But differently from Stewart's approach, we do not admit reasoning about refutations.

Instead of an empty conclusion after the application of the μ-rule and before the application of the $[-]$-rule, we use the logical absurdity (which in Nuprl is defined on the empty data type `void`, thus corresponding to the "emptiness" of the active position). Therefore we do not need additional rules for the introduction or elimination of $\bot$. This approach also coincides with Ong's $\lambda\mu$theory. Furthermore Pym and Ritter [PR04] propose a version of $\lambda\mu\nu$ which also uses $\bot$ instead of an empty active position, and omitting the additional $\bot$ rules (we will refer to this version as $\lambda\mu\nu_{\bot}$, see fig. 4.2).

Another difference of $\lambda\mu$PRL and $\lambda\mu\nu$ is the handling of the disjunction. Whereas in $\lambda\mu\nu$ $\vee I$ moves the rightmost formula of the disjunction to the context and $\vee E$ does the reverse, $\lambda\mu$PRL uses the "constructive" $\vee$ rules corresponding to Nuprl's sum type. Therefore the choice between the two disjuncts is not locally represented, but has to be taken care of before choosing one disjunct (by "storing" the disjunction within the μ-context). This treatment of the disjunction can also be found in another variant of $\lambda\mu\nu$, presented by Pym and Ritter in [PR01] as $\lambda\mu\oplus$. We will use a combination of $\lambda\mu\nu_{\bot}$ and $\lambda\mu\oplus$ in section 4.2 to prove the consistency of $\lambda\mu$PRL.

Corresponding to both Nuprl's terms as well as Pym's and Ritter's $\lambda\mu\nu$ in [PR04], the terms of $\lambda\mu$PRL are not typed.

The sequents of $\lambda\mu$PRL can be viewed as term assignment judgements: $\Gamma \vdash T$ ext t means the type T is inhabited by the term t. Γ ranges over both λ and μ-contexts. Accordingly there are two kinds of variables, the λ- and the μ-variables, ranging over two distinct alphabets. λ-variables are declared by $x : A, y : B, \ldots$ and bound by the λ-abstraction $\lambda x.t$. The declaration of μ-variables is syntactically distinguished by an encapsulation $\{\{A^\alpha\}\}$, $\{\{B^\beta\}\}, \ldots$. Their binding operator is the μ-abstraction as in $\mu\alpha.t$.

The Types ranging over $A, B, C, \ldots$. are built up by the constructors $\supset, \wedge, \vee$ from a set of atomic types and the distinguished absurdity type $\bot$. The Negation $\neg A$ is expressed by $A \supset \bot$.
Terms of $\lambda\mu$PRL are build up by the following grammar:

$$t ::== x \mid \lambda x.t \mid \mu x.t \mid (t\ t) \mid [x]t \mid \langle t, t\rangle \mid let\ \langle x, x\rangle = t\ in\ t \mid$$
$$inl(t) \mid inr(t) \mid case\ t\ of\ inl(x) : t \mid inr(x) : t$$

The rules of the calculus $\lambda\mu$PRL can be found in figure 4.1. Some of the rules need additional parameters for technical reasons:

- The elimination- and `abort`-rules as well as the `hypothesis`-rule need the position of the formula to work on denoted by $\$i$.
- The `mu`-rules and `lambdaI` need the name of the variable to be declared (which must be previously undeclared)

4.2 Consistency of $\lambda\mu$PRL

The consistency of $\lambda\mu$PRL can be established based on the consistency of the $\lambda\mu\nu$-calculus. $\lambda\mu$'s consistency is shown by Pym and Ritter in [PR04, PR01] by providing both a categorical and a games model. Their categorical model involves fibred bi-cartesian closed categories which have sufficient structure to not collapse to boolean algebras when modeling classical logic.

Since $\lambda\mu$PRL does not provide the possibility of an ,,empty" conclusion, we will first show the equivalence of $\lambda\mu\nu$ to its variant $\lambda\mu\oplus_\bot$ (a combination of $\lambda\mu\nu_\bot$ and $\lambda\mu\oplus$) and then use $\lambda\mu\oplus_\bot$ to proof the consistency of $\lambda\mu$PRL by a mutual simulation. Different from $\lambda\mu\nu$, $\lambda\mu\oplus_\bot$ does not contain an empty conclusion either and like $\lambda\mu$PRL uses $\bot$ as type instead, thus rendering the additional rules $\bot I$ and $\bot E$ redundant and omitting them. The other difference is their treatment of the disjunction. In $\lambda\mu\nu$ the passive context is involved in the $\vee$-rules and the right-hand side disjunct is automatically moved to or from the context; $\lambda\mu\oplus_\bot$ supports the intuitionistic view of disjunction, demanding a decision for one of the disjuncts in the associated introduction rule.

Pym and Ritter claim the variants $\lambda\mu\nu_\bot$ and $\lambda\mu\oplus$ to be equivalent to $\lambda\mu\nu$ in the sense that there exist translations between the two calculi. We will show the same is valid for the combination of these two calculi, $\lambda\mu\oplus_\bot$. Based on this prerequisite, we will establish the consistency of $\lambda\mu$PRL by a mutual simulation with $\lambda\mu\oplus_\bot$.

G, $x:T$, H $\vdash T$ ext x BY hypothesis $\$i$ (no subgoals)	
H $\vdash A \supset B$ ext $\lambda x.b$ BY lambdaI x H, $x{:}A \vdash B$ ext b	G, $pf{:}A \supset B$, H $\vdash$ C ext $b[pf\ a/y]$ BY functionE $\$i$ G, $pf{:}A \supset B$, H $\vdash$ A ext a G, $y{:}B$, H $\vdash$ C ext b
H $\vdash B$ ext $\mu\_\alpha.b$ BY mu1 α H , $\{\{B^\alpha\}\} \vdash \bot$ ext b	G ,$\{\{B^\alpha\}\}$, H$\vdash \bot$ ext $[\alpha]\ b$ BY abort1 $\$i$ G , H $\vdash B$ ext b
H $\vdash B$ ext $\mu\_\alpha.b$ BY mu2 α H $\vdash \bot$ ext b	G ,$\{\{B^\alpha\}\}$, H $\vdash \bot$ ext $[\alpha]\ b$ BY abort2 $\$i$ G , $\{\{B^\alpha\}\}$, H $\vdash B$ ext b
H $\vdash A \wedge B$ ext $< a, b >$ BY andI H $\vdash A$ ext a H $\vdash B$ ext b	G, $z{:}A \wedge B$, H $\vdash$ C ext let $< a, b >$ $= z$ in u BY andE $\$i$ H , $a : A$, $b : B$, G $\vdash$ C ext u
H $\vdash A \vee B$ ext inl(a) BY orI11 H $\vdash A$ ext a H $\vdash A \wedge B$ ext inr(b) BY orI2 H $\vdash B$ ext b	G, $z{:}A \vee B$, H $\vdash$ C ext case z of inl(a) => u | inr(b) => v BY orE $\$i$ H , $a : A$, G $\vdash$ C ext u H , $b : B$, G $\vdash$ C ext v

Figure 4.1: The calculus λμPRL

$\lambda\mu\nu_\bot$: terms $t ::== x \mid \lambda x.t \mid (t\ t) \mid \mu\alpha.t \mid [\alpha]\,t \mid \langle t,t\rangle \mid \pi(t) \mid \pi`(t) \mid \langle\beta\rangle\,t \mid \nu\beta.t \mid \langle\bot\rangle\,t \mid \nu\bot.t$

The rules for $\supset$,$\wedge$ and $\vee$ as well as Ax are the same as in $\lambda\mu\nu$.

$$\frac{\Gamma \vdash t : \bot, \phi^\alpha, \Delta}{\Gamma \vdash \mu\alpha.t : \phi, \Delta}\ \mu \qquad \frac{\Gamma \vdash t : \phi, \Delta}{\Gamma \vdash [\alpha]\,t : \bot, \phi^\alpha, \Delta}\ [\_]$$

$$\frac{\Gamma \vdash t : \bot, \Delta}{\Gamma \vdash \mu\alpha.t : \phi, \Delta}\ \mu \qquad \frac{\Gamma \vdash t : \phi, \phi^\alpha, \Delta}{\Gamma \vdash [\alpha]\,t : \bot, \phi^\alpha, \Delta}\ [\_]$$

Figure 4.2: The calculus $\lambda\mu\nu_\bot$

$\lambda\mu\oplus$: terms $t ::== x \mid \lambda x : \phi.t \mid (t\ t) \mid \mu\alpha.t \mid \mu\bot.t \mid [\alpha]\,t \mid [\bot]\,t \mid \langle t,t\rangle \mid \pi(t) \mid \pi`(t) \mid \mathtt{in_1}(t) \mid \mathtt{in_2}(t) \mid \mathtt{case}\ t\ \mathtt{of}\ \mathtt{in_1}(x) \Rightarrow t\ \mathtt{or}\ \mathtt{in_2}(y) \Rightarrow t$

The rules for $\supset$ and $\wedge$ as well as Ax, μ and $[-]$ are the same as in $\lambda\mu\nu$.

$$\frac{\Gamma \vdash t : \phi_i, \Delta}{\Gamma \vdash \mathtt{in}_i(t) : \phi_1 \vee \phi_2, \Delta}\ \vee^+ I \qquad i \in \{1,2\}$$

$$\frac{\Gamma \vdash t : \phi \vee \psi, \Delta \qquad \Gamma, x : \phi \vdash u : \chi, \Delta \qquad \Gamma, y : \psi \vdash v : \chi, \Delta}{\Gamma \vdash \mathtt{case}\ t\ \mathtt{of}\ \mathtt{in_1}(x) \Rightarrow u\ \mathtt{or}\ \mathtt{in_2}(y) \Rightarrow v : \chi, \Delta}\ \vee^+ E$$

Figure 4.3: The calculus $\lambda\mu\oplus$

$\lambda\mu\oplus_\bot$: terms $t ::== x \mid \lambda x.t \mid (t\ t) \mid \mu\alpha.t \mid [\alpha]\,t \mid \langle t,t\rangle \mid \pi(t) \mid \pi`(t) \mid \mathtt{in_1}(t) \mid \mathtt{in_2}(t) \mid \mathtt{case}\ t\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t\ \mathtt{or}\ \mathtt{inr}(y) \Rightarrow t$

Consists of the rules for $\supset$ and $\wedge$ and the rule Ax of $\lambda\mu\nu$;
the rules μ and $[-]$ of $\lambda\mu\nu_\bot$, and the rules $\vee I$ and $\vee E$ of $\lambda\mu\oplus$.

Figure 4.4: The calculus $\lambda\mu\oplus_\bot$

Lemma 4.1. *There exists a translation between the calculi $\lambda\mu\nu$ and $\lambda\mu\oplus_\perp$ which relates every term of $\lambda\mu\nu$ to an equivalent term in $\lambda\mu\oplus_\perp$ and reversely.*

Proof. As mentioned above, $\lambda\mu\nu$ (fig.2.5) and $\lambda\mu\oplus_\perp$ (fig.4.4) merely differ in their treatment of disjunction and logical absurdity. $\lambda\mu\nu$ contains two additional rules $\perp I$ and $\perp E$ which insert the absurdity $\perp$ into the active part of the conclusion, respectively remove it, resulting in an empty active type. As a consequence, the active position has to be empty to fetch a formula from the context and remains empty after moving a formula to the context. In $\lambda\mu\nu_\perp$, on the contrary, there is no empty active type of the conclusion, but instead the type $\perp$ is assigned to the term. Therefore the rules for $\perp$ are subsumed by the rules μ and $[-]$. Each application of one of these rules in $\lambda\mu\oplus_\perp$ can however be achieved by a sequence of rule applications in $\lambda\mu\nu$. Since in most cases this is quite obvious, we simply provide the corresponding rule applications and the translations for the accompanying terms. Otherwise we show how to derive the conclusion of the rule from the premiss(es).

$\Rightarrow$) Terms of $\lambda\mu\oplus_\perp$ translated into $\lambda\mu\nu$

1. μ-rule (both variants): Realized by a sequence of μ, $\perp E$. Thus we get the following translation:

$$[[\mu\alpha.t]]_{\lambda\mu\oplus_\perp} \equiv [[\mu\alpha.[\perp]t]]_{\lambda\mu\nu}$$

2. $[-]$ (both variants): Sequence of $\perp I$, $[-]$ resulting in

$$[[[\alpha]t]]_{\lambda\mu\oplus_\perp} \equiv [[\mu\perp.[\alpha]t]]_{\lambda\mu\nu}$$

3. $\vee I_1$: $[-]$, μ, $[-]$, μ, $\vee I$, therefore

$$[[\mathtt{in_1}(t)]]_{\lambda\mu\oplus_\perp} \equiv [[\nu\beta.\mu\alpha.[\beta]\mu\beta.[\alpha]t]]_{\lambda\mu\nu}$$

4. $\vee I_2$: $[-]$, μ, $\vee I$ resulting in

$$[[\mathtt{in_2}(t)]]_{\lambda\mu\oplus_\perp} \equiv [[\nu\beta\mu\alpha.[\beta].t]]_{\lambda\mu\nu}$$

5. $\vee E$: In this case we will give a complete derivation in $\lambda\mu\nu$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : \phi \vdash u : \chi, \Delta}{\Gamma \vdash \lambda x.u : \phi \supset \chi, \Delta}\supset I}{\Gamma \vdash [\alpha]\lambda x.u : \ \phi \supset \chi^\alpha, \Delta}[-]}{\Gamma \vdash \mu\beta.[\alpha]\lambda x.u : \psi, \phi \supset \chi^\alpha, \Delta}\mu}{\Gamma \vdash [\beta]\mu\beta.[\alpha]\lambda x.u : \ \psi^\beta, \phi \supset \chi^\alpha, \Delta}[-]$$

$$\dfrac{\dfrac{\Gamma \vdash t : \phi \vee \psi, \Delta}{\Gamma \vdash \langle\beta\rangle\, t : \phi, \psi^\beta, \Delta}\vee E \qquad \dfrac{\Gamma \vdash [\beta]\mu\beta.[\alpha]\lambda x.u : \ \psi^\beta, \phi \supset \chi^\alpha, \Delta}{\Gamma \vdash \mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u : \phi \supset \chi, \psi^\beta, \Delta}\mu}{\Gamma \vdash (\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t) : \chi, \psi^\beta, \Delta}\supset E$$

$$\dfrac{\dfrac{\Gamma \vdash (\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t) : \chi, \psi^\beta, \Delta}{\Gamma \vdash [\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t) : \ \chi^\epsilon, \psi^\beta, \Delta}[-]}{\Gamma \vdash \mu\beta.[\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t) : \psi, \chi^\epsilon, \Delta}\mu$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, y : \psi \vdash v : \chi, \Delta}{\Gamma \vdash \lambda y.v : \psi \supset \chi, \Delta}\supset I}{\Gamma \vdash [\gamma]\lambda y.v : \ \psi \supset \chi^\gamma, \Delta}[-]}{\Gamma \vdash \mu\epsilon.[\gamma]\lambda y.v : \chi, \psi \supset \chi^\gamma, \Delta}\mu}{\dfrac{\Gamma \vdash [\epsilon]\mu\epsilon.[\gamma]\lambda y.v : \ \chi^\epsilon, \psi \supset \chi^\gamma, \Delta}{\Gamma \vdash \mu\gamma.[\epsilon]\mu\epsilon.[\gamma]\lambda y.v : \psi \supset \chi, \Delta}\mu}[-]$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mu\beta.[\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t) : \psi, \chi^\epsilon, \Delta \qquad \Gamma \vdash \mu\gamma.[\epsilon]\mu\epsilon.[\gamma]\lambda y.v : \psi \supset \chi, \Delta}{\Gamma \vdash (\mu\gamma.[\epsilon]\mu\epsilon.[\gamma]\lambda y.v)(\mu\beta.[\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t)) : \chi, \chi^\epsilon, \Delta}\supset E}{\Gamma \vdash [\epsilon](\mu\gamma.[\epsilon]\mu\epsilon.[\gamma]\lambda y.v)(\mu\beta.[\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t)) : \ \chi^\epsilon, \Delta}[-]}{\Gamma \vdash \mu\epsilon.[\epsilon](\mu\gamma.[\epsilon]\mu\epsilon.[\gamma]\lambda y.v)(\mu\beta.[\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t)) : \ \chi, \Delta}\mu$$

$$[[\texttt{case}\ t\ \texttt{of}\ \texttt{in}_1(x) \Rightarrow u\ \texttt{or}\ \texttt{in}_2(y) \Rightarrow v]]_{\lambda\mu\oplus_\perp} \equiv$$
$$[[\mu\epsilon.[\epsilon](\mu\gamma.[\epsilon]\mu\epsilon.[\gamma]\lambda y.v)(\mu\beta.[\epsilon](\mu\alpha.[\beta]\mu\beta.[\alpha]\lambda x.u)(\langle\beta\rangle\, t))]]_{\lambda\mu\nu}$$

6. Since apart from the above all rules of $\lambda\mu\oplus_\perp$ are identical to the ones of $\lambda\mu\nu$, so are the corresponding terms.

$\Leftarrow$) Terms of $\lambda\mu\nu$ translated into $\lambda\mu\oplus_\perp$

1. μ-rule (both variants): It is not possible to derive the syntactically identical rule, since there is no "empty" conclusion in $\lambda\mu\oplus_\perp$. However since the μ-rule of $\lambda\mu\oplus_\perp$ subsumes the rules μ and $\perp$ of $\lambda\mu\nu$ it is possible to perform a case split on the term t. There are only two rules which could have been applied before an application of the μ-rule in $\lambda\mu\nu$ which are $[-]$ and $\perp E$:
   (a) $[-]$: An application of $[-]$ followed by μ in $\lambda\mu\nu$ is realized by the corresponding rules in $\lambda\mu\oplus$, thus:
   $$[[\mu\alpha.[\beta]t]_{\lambda\mu\nu} \equiv [[\mu\alpha.[\beta]t]]_{\lambda\mu\oplus_\perp}$$
   (b) $\perp E$: An application of $\perp E$ followed by μ in $\lambda\mu\nu$ is realized by an application of the μ-rule in $\lambda\mu\oplus$, thus:
   $$[[\mu\alpha.[\perp]t]_{\lambda\mu\nu} \equiv [[\mu\alpha.t]]_{\lambda\mu\oplus_\perp}$$
2. $[-]$ (both variants): Out of the same reason as above, we need to perform a case split on the environment of the term $[\alpha]t$. There are only two rules which could be applied after an application of the $[-]$-rule in $\lambda\mu\nu$ which are μ and $\perp I$:
   (a) μ: An application of $[-]$ followed by μ in $\lambda\mu\nu$ is realized by the corresponding rules in $\lambda\mu\oplus$, thus as already established in case (1a):
   $$[[\mu\alpha.[\beta]t]_{\lambda\mu\nu} \equiv [[\mu\alpha.[\beta]t]]_{\lambda\mu\oplus_\perp}$$
   (b) $\perp E$: An application of $[-]$ followed by $\perp I$ in $\lambda\mu\nu$ is realized by an application of the $[-]$-rule in $\lambda\mu\oplus$, thus:
   $$[[\mu\perp.[\alpha]t]_{\lambda\mu\nu} \equiv [[\ [\alpha]t]]_{\lambda\mu\oplus_\perp}$$

3. $\vee I$: Can be derived by the sequence $\vee I_1$, $[-]$,μ, $\vee I_2$, $[-]$, μ in $\lambda\mu\oplus$:

$$[[\nu\beta.t]]_{\lambda\mu\nu} \equiv [[\mu\alpha[\alpha]\mathtt{in_2}(\mu\beta.[\alpha]\mathtt{in_1}(t))]]_{\lambda\mu\oplus_\perp}$$

4. $\vee I_\perp$: Is derived by an application of $\vee I_1$ in $\lambda\mu\oplus_\perp$:

$$[[\nu\perp.t]]_{\lambda\mu\nu} \equiv [[\mathtt{in_1}(t)]]_{\lambda\mu\oplus_\perp}$$

5. $\vee E$: In this case we will give a complete derivation in $\lambda\mu\oplus_\perp$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash t : \phi \vee \psi, \Delta}{\Gamma \vdash [\alpha]t : \perp, \phi \vee \psi^\alpha, \Delta}\,[-]}{\Gamma \vdash \mu\beta.[\alpha]t : \psi, \phi \vee \psi^\alpha, \Delta}\,\mu}{\Gamma \vdash [\beta]\mu\beta.[\alpha]t : \perp, \psi^\beta, \phi \vee \psi^\alpha, \Delta}\,[-]}{\Gamma \vdash \mu\alpha.[\beta]\mu\beta.[\alpha]t : \phi \vee \psi, \psi^\beta, \Delta}\,\mu \quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : \phi \vdash x : \phi, \Delta}{\Gamma, x : \phi \vdash [\gamma]x : \perp, \phi^\gamma, \Delta}\,[-]}{\Gamma, x : \phi \vdash \mu\beta.[\gamma]x : \psi, \phi^\gamma, \Delta}\,\mu}{\Gamma, x : \phi \vdash [\beta]\mu\beta.[\gamma]x : \perp, \psi^\beta, \phi^\gamma, \Delta}\,[-]}{\Gamma, x : \phi \vdash \mu\gamma.[\beta]\mu\beta.[\gamma]x : \phi, \psi^\beta, \Delta}\,\mu \quad \dfrac{\dfrac{\dfrac{\Gamma, y : \psi \vdash y : \psi, \Delta}{\Gamma, y : \psi \vdash [\beta]y : \perp, \psi^\beta, \Delta}\,[-]}{\Gamma, y : \psi \vdash \mu\delta.[\beta]y : \phi, \psi^\beta, \Delta}\,\mu}{\Gamma \vdash \mathtt{case}\ \mu\alpha.[\beta]\mu\beta.[\alpha]t\ \mathtt{of}\ \mathtt{in_1}(x) \Rightarrow \mu\gamma.[\beta]\mu\beta.[\gamma]x\ \mathtt{or}\ \mathtt{in_2}(y) \Rightarrow \mu\delta.[\beta]y : \phi, \psi^\beta, \Delta}\,\vee E$$

Resulting in:

$$[[\langle\beta\rangle\, t]]_{\lambda\mu\nu} \equiv [[\mathtt{case}\ \mu\alpha.[\beta]\mu\beta.[\alpha]t\ \mathtt{of}\ \mathtt{in_1}(x) \Rightarrow \mu\gamma.[\beta]\mu\beta.[\gamma]x\ \mathtt{or}\ \mathtt{in_2}(y) \Rightarrow \mu\delta.[\beta]y]]_{\lambda\mu\oplus_\perp}$$

6. $\vee E_\perp$: The derivation in $\lambda\mu\oplus_\perp$ is:

$$\dfrac{\Gamma \vdash t : \phi \vee \perp, \Delta \qquad \dfrac{\Gamma, x : \phi \vdash x : \phi, \Delta}{}\,Ax \qquad \dfrac{\dfrac{\Gamma, y : \perp \vdash y : \perp, \Delta}{\Gamma, y : \perp \vdash \mu\alpha.y : \phi, \Delta}\,\mu}{}\,Ax}{\Gamma \vdash \mathtt{case}\ t\ \mathtt{of}\ \mathtt{in_1}(x) \Rightarrow x\ \mathtt{or}\ \mathtt{in_2}(y) \Rightarrow \mu\alpha.y : \phi, \Delta}\,\vee E$$

$$[[\langle\perp\rangle\, t]]_{\lambda\mu\nu} \equiv [[\mathtt{case}\ t\ \mathtt{of}\ \mathtt{in_1}(x) \Rightarrow x\ \mathtt{or}\ \mathtt{in_2}(y) \Rightarrow \mu\alpha.y]]_{\lambda\mu\oplus_\perp}$$

□

Remark Essentially, apart from the obviously different treatment of disjunction, the main difference between terms of $\lambda\mu\nu$ and $\lambda\mu\oplus_\perp$ is the following: In $\lambda\mu\nu$ a term $[\alpha]t$ can only occur as sub-term of a μ-binding. If during the proof we had an introduction of $\perp$ it is signalled by $\mu\perp$ in the term whereas in $\lambda\mu\oplus_\perp$ an introduction of absurdity is "hidden" in a singular occurrence of a $[-]$-term.

Using Lemma 4.1 we can now proceed to proving:

Proposition 4.2. *A formula ϕ is provable in $\lambda\mu$PRL within the λ-context of $x_1 : H_1, \ldots, x_n : H_n$ and the μ-context of $\alpha_1 : \Delta_1, \ldots, \alpha_n : \Delta_n$, iff $x_1 : H_1, \ldots, x_n : H_n \vdash \phi, \Delta_1^{\alpha_1}, \ldots, \Delta_n^{\alpha_n}$ is provable in $\lambda\mu\nu$.*

Proof. We have proven the existence of a translation between $\lambda\mu\nu$ and $\lambda\mu\oplus_\perp$ as lemma 4.1. We continue by mutually simulating the rules of $\lambda\mu\oplus_\perp$ and $\lambda\mu$PRL. The comma on the left-hand side of the sequent is interpreted as $\wedge$, the $\vdash$ as $\supset$; context formulae are treated as context formulae, adjusted to the syntax of the calculus in question. The syntactic representation of the context is irrelevant in the sense that the context formula

are distinguished from the rest of the sequent in both calculi and can only be accessed by the specific exchange rules.

To focus on the relevant part of the rules, we prove the "core" of the rules. This means we neglect the side formulae (usually denoted by Γ and Δ) which are not required for or altered by the the application of the respective rule.

1. $\supset I$ and `lambdaI`: Trivial, since the only difference between the rules is the syntactic position of the μ-context.Moreover the semantics of the $\vdash$ delimiter coincides with the semantics of the implication, thus the proof of the rule results in an identity map.

   Simulation of $\supset I$ in $\lambda\mu$PRL:

   $\vdash (\phi \supset \psi) \supset (\phi \supset \psi)$ **ext** λx.x
   by lambdaI x
   $\lfloor\_x : \phi \supset \psi \vdash \phi \supset \psi$ **ext** x
   by hypothesis 1

   Simulation of `lambdaI` in $\lambda\mu\oplus_\perp$:

   $$\dfrac{\dfrac{}{x : A \supset B \vdash x : A \supset B}\,Ax}{\vdash \lambda x.x : (A \supset B) \supset (A \supset B)}\,\supset I$$

2. $\supset E$ and `functionE`:

   Simulation of $\supset E$ in $\lambda\mu$PRL:

   $\vdash ((\phi \supset \psi) \wedge \phi) \supset \psi$ **ext** λx.**let** $\langle$f, s$\rangle$ = x **in** (f s)
   by lambdaI x
   $\lfloor\_x : (\phi \supset \psi) \wedge \phi \vdash \psi$ **ext** **let** $\langle$f, s$\rangle$ = x **in** (f s)
   by andE 1
   $\lfloor\_f : \phi \supset \psi, s : \phi \vdash \psi$ **ext** t [(f s)/t]
   by functionE 1
   $\lfloor\_s : \phi \vdash \phi$ **ext** s
   by hypothesis 1
   $\lfloor\_t : \psi, s : \phi \vdash \psi$ **ext** t
   by hypothesis 1

   Simulation of `functionE` in $\lambda\mu\oplus_\perp$:

   $\xi \equiv A \wedge (B \supset C)$ and $\chi \equiv A \supset B$

   $$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \xi, y : \chi \vdash x : A \wedge (B \supset C)}\,Ax}{x : \xi, y : \chi \vdash \pi'(x) : B \supset C}\,\wedge E2 \quad \dfrac{\dfrac{}{x : \xi, y : \chi \vdash y : A \supset B}\,Ax \quad \dfrac{\dfrac{}{x : \xi, y : \chi \vdash x : A \wedge (B \supset C)}\,Ax}{x : \xi, y : \chi \vdash \pi(x) : A}\,\wedge E1}{x : \xi, y : \chi \vdash (y\ \pi(x))) : B}\,\supset E}{x : A \wedge (B \supset C), y : A \supset B \vdash (\pi'(x)\ (y\ \pi(x))) : C}\,\supset E}{x : A \wedge (B \supset C) \vdash \lambda y.(\pi'(x)\ (y\ \pi(x))) : (A \supset B) \supset C}\,\supset I}{\vdash \lambda x.\lambda y.(\pi'(x)\ (y\ \pi(x))) : (A \wedge (B \supset C)) \supset ((A \supset B) \supset C)}\,\supset I$$

3. $\wedge I$ and andI: Trivial since the rules are identical but for their syntactic layout. Moreover the semantics of the comma delimiter in the hypotheses coincides with the semantics of the connective $\wedge$, thus the proof of the rule results in an identity map.

   Simulation of $\wedge I$ in $\lambda\mu$PRL:

   $\vdash (\phi \wedge \psi) \supset (\phi \wedge \psi)$ **ext** λx.x
   by lambdaI x
   $|\_x : (\phi \wedge \psi) \vdash \phi \wedge \psi$ **ext** x
   by hypothesis 1

   Simulation of `andI` in $\lambda\mu\oplus_\perp$:

$$\dfrac{\dfrac{}{x : A \wedge B \vdash x : A \wedge B}\,{\scriptstyle Ax}}{\vdash \lambda x.x : (A \wedge B) \supset (A \wedge B)}\,{\scriptstyle \supset I}$$

4. $\wedge E_1$ and `andE`$_1$:

   Simulation of $\wedge E_1$ in $\lambda\mu$PRL:

   $\vdash (\phi \wedge \psi) \supset \phi$ **ext** λx.let $\langle$s,t$\rangle$ = x in s
   by lambdaI x
   $|\_x : (\phi \wedge \psi) \vdash \phi$ **ext** let $\langle$s,t$\rangle$ = x in s
   by andE 1
   $|\_s : \phi, t : \psi \vdash \phi$ **ext** s
   by hypothesis 1

   Simulation of `andE` in $\lambda\mu\oplus_\perp$:

$$\dfrac{\dfrac{}{x : (A \wedge B) \supset C \vdash x : (A \wedge B) \supset C}\,{\scriptstyle Ax}}{\vdash \lambda x.x : ((A \wedge B) \supset C) \supset ((A \wedge B) \supset C)}\,{\scriptstyle \supset I}$$

5. $\wedge E_2$ and `andE`$_2$: Analogously to $\wedge E_1$ and `andE`$_1$. The corresponding terms are λx.let $\langle$s,t$\rangle$ = x in t in $\lambda\mu$PRL and $\lambda x.x$ again in $\lambda\mu\oplus_\perp$.

6. $\vee I_i, i \in \{1, 2\}$ and orI

   Simulation of $\vee I_1$ in $\lambda\mu$PRL:

   $\vdash \phi \supset \phi \vee \psi$ **ext** λx.inl(x)
   by lambdaI x
   $|\_x : \phi \vdash \phi \vee \psi$ **ext** inl(x)
   by orI1
   $|\_x : \phi \vdash \phi$ **ext** x
   by hypothesis 1

   Simulation of $\vee I_2$ in $\lambda\mu$PRL:

   $\vdash \psi \supset \phi \vee \psi$ **ext** λx.inr(x)
   by lambdaI x
   $|\_x : \psi \vdash \phi \vee \psi$ **ext** inr(x)
   by orI2
   $|\_x : \psi \vdash \phi$ **ext** x
   by hypothesis 1

Simulation of $\texttt{orI}_1$ and $\texttt{orI}_2$ in $\lambda\mu\oplus_\perp$:

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,Ax}{x : A \vdash \texttt{in}_1(x) : A \vee B}\,\vee I_1}{\vdash \lambda x.\texttt{in}_1(x) : A \supset (A \vee B)}\,\supset I \qquad \dfrac{\dfrac{\dfrac{}{x : B \vdash x : B}\,Ax}{x : B \vdash \texttt{in}_2(x) : A \vee B}\,\vee I_2}{\vdash \lambda x.\texttt{in}_2(x) : B \supset (A \vee B)}\,\supset I$$

7. $\vee E$ and orE

Simulation of $\vee E$ in $\lambda\mu$PRL:

$\vdash ((\phi \vee \psi) \wedge (\phi \supset \chi) \wedge (\psi \supset \chi)) \supset \chi$	**ext** λx.let $<$x,v$>=$ w in
by lambdaI x	let $<$x,y$>=$ v in case x of ...
$\mid\!\_\ w : (\phi \vee \psi) \wedge (\phi \supset \chi) \wedge (\psi \supset \chi) \vdash \chi$	**ext** let $<$x,v$>=$ w in
by andE 1	let $<$x,y$>=$ v in case x of ...
$\mid\!\_\ x : \phi \vee \psi, v : (\phi \supset \chi) \wedge (\psi \supset \chi) \vdash \chi$	**ext** let $<$x,y$>=$ v in case x of ...
by andE 2	
$\mid\!\_\ x : \phi \vee \psi, y : \phi \supset \chi, z : \psi \supset \chi \vdash \chi$	**ext** case x of inl(s) $\Rightarrow$ (y s)$\mid$
by orE 1	inr(t) $\Rightarrow$ (y t)
$\mid\!\_\ s : \phi, y : \phi \supset \chi, z : \psi \supset \chi \vdash \chi$	**ext** (z s)
by functionE 2	
$\mid\ \mid\!\_\ s : \phi, y : \phi \supset \chi, z : \psi \supset \chi \vdash \phi$	**ext** s
by hypothesis 1	
$\mid\ \mid\!\_\ s : \phi, r : \chi, z : \psi \supset \chi \vdash \chi$	**ext** r
by hypothesis 2	
$\mid\!\_\ t : \psi, y : \phi \supset \chi, z : \psi \supset \chi \vdash \chi$	**ext** (y t)
by functionE 3	
$\mid\!\_\ t : \psi, y : \phi \supset \chi, z : \psi \supset \chi \vdash \psi$	**ext** t
by hypothesis 1	
$\mid\!\_\ t : \psi, u : \chi, z : \psi \supset \chi \vdash \chi$	**ext** u
by hypothesis 2	

Simulation of orE in $\lambda\mu\oplus_\perp$:

$\phi \equiv (A \supset C) \wedge (B \supset C)$
$\psi \equiv A \vee B$
$\Gamma_A \equiv x : \phi, y : \psi, u : A$
$\Gamma_B \equiv x : \phi, y : \psi, v : B$

$$\dfrac{\dfrac{\dfrac{}{x : \phi, y : \psi \vdash y : (A \vee B)}\,Ax \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_A \vdash x : \phi}\,Ax}{\Gamma_A \vdash \pi(x) : A \supset C}\,\wedge E \quad \dfrac{}{\Gamma_A \vdash u : A}\,Ax}{\Gamma_A \vdash \pi(x)\ u : C}\,\supset E \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_B \vdash x : \phi}\,Ax}{\Gamma_B \vdash \pi'(x) : B \supset C}\,\wedge E \quad \dfrac{}{\Gamma_B \vdash v : B}\,Ax}{\Gamma_B \vdash \pi'(x)\ v : C}\,\supset E}{x : (A \supset C) \wedge (B \supset C), y : (A \vee B) \vdash \texttt{case}\ y\ \texttt{of}\ \texttt{in}_1(u) \Rightarrow \pi(x)\ u\ \texttt{or}\ \texttt{in}_2(v) \Rightarrow \pi'(x)\ v : C}\,\vee E}{x : (A \supset C) \wedge (B \supset C) \vdash \lambda y.\texttt{case}\ y\ \texttt{of}\ \texttt{in}_1(u) \Rightarrow \pi(x)\ u\ \texttt{or}\ \texttt{in}_2(v) \Rightarrow \pi'(x)\ v : (A \vee B) \supset C}\,\supset I}{\vdash \lambda x.\lambda y.\texttt{case}\ y\ \texttt{of}\ \texttt{in}_1(u) \Rightarrow \pi(x)\ u\ \texttt{or}\ \texttt{in}_2(v) \Rightarrow \pi'(x)\ v : ((A \supset C) \wedge (B \supset C)) \supset (A \vee B) \supset C}\,\supset I$$

Remark The simulation of the exchange rules μ and $[-]$ needs further explanation. On the one hand, the equivalence is quite obvious, since in both calculi these rules fulfil the same function of moving formulae to and from the context. However in the simulation we have to deal with the manipulation of the passive context, so on the other hand some thought is required how to handle the alteration of the context. The question is whether or not we may use the context formula in proving the rule as a theorem.

Not surprisingly, the answer depends on the rule in question, with the μ-rule playing the role of $\bot$ elimination while $[-]$ is the non-constructive rule of $\bot$ introduction. Accordingly $[-]$ is the rule in which the succedent in the conclusion is only provable from the hypotheses in the presence of the matching context formula. Opposed to this, the rule μ is basically independent form the context formulae (which is most obvious in the second rule μ). Thus in the simulation we may use the context formula as a prerequisite in the proof of the rule $[-]$ but not in the proof of μ.

The reader may note that if we treated the context formulae as part of a disjunction in the conclusion (or in any other way as part of the goal formula), we would fail to reflect the distinct syntactic position of the context.

The two variants of each of the exchange rules do not differ in their simulation, since the second instances of the rules are merely structural rules, i.e. contraction and weakening.

8. μ and $\mathtt{mu}i$, $i \in \{1, 2\}$

Simulation of μ in $\lambda\mu$PRL:

$\vdash \bot \supset \phi$	**ext** $\lambda\mathtt{x}.\mu\alpha.\mathtt{x}$
by lambdaI x	
$\mid\_\ x : \bot \vdash \phi$	**ext** $\mu\alpha.\mathtt{x}$
by mu1 alpha	
$\mid\_\ x : \bot, \{\{\phi^{\alpha}\}\} \vdash \bot$	**ext** $\mathtt{x}$
by hypothesis 1	

Simulation of $\mathtt{mu}$ in $\lambda\mu\oplus_{\bot}$:

$$\frac{}{x : \bot \vdash x : \bot}\ Ax$$

$$\frac{}{x : \bot \vdash \mu\alpha.x : C}\ \mu$$

$$\frac{}{\vdash \lambda x.\mu\alpha.x : \bot \supset C}\ \supset I$$

9. $[-]$ and $\mathtt{abort}i$, $i \in \{1,2\}$

   Simulation of $[-]$ in $\lambda\mu$PRL:

   $\{\{\phi^\alpha\}\} \vdash \phi \supset \bot$ **ext** $\lambda\mathtt{x}.[\alpha]\mathtt{x}$
   by lambdaI x
   $|\_\ x : \phi, \{\{\phi^\alpha\}\} \vdash \bot$ **ext** $[\alpha]\mathtt{x}$
   by abort1 2
   $|\_\ x : \phi \vdash \phi$ **ext** $\mathtt{x}$
   by hypothesis 1

   Simulation of $\mathtt{mu}$ in $\lambda\mu\oplus_\bot$:

$$\dfrac{\dfrac{\dfrac{}{x : C \vdash x : C}\ Ax}{x : C \vdash [\alpha]x : \bot}\ [-]}{\vdash \lambda x.[\alpha]x : C \supset \bot, C^\alpha}\ \supset I$$

□

Chapter 5

Conversion Theory

When we established the soundness of $\lambda\mu$PRL in the last chapter, we postponed the crucial part of an accompanying conversion theory. A reduction relation is both important from a proof-theoretic and from a computational viewpoint: On the on hand, reduction rules define the extensional equality of proofs and on the other hand, the process of reduction (and thus cut elimination) represents computation.

In the scope of this work, our main interest concerning reduction is a confluent cut elimination procedure, thereby providing a sound basis for equality reasoning about $\lambda\mu$PRL-terms.

This chapter consists of two parts. The first section reflects on the conversion theory of different $\lambda\mu$-calculi in relation to $\lambda\mu$PRL. It also contains definitions of β-,μ-,δ and π-reduction for $\lambda\mu$PRL, but for our current purpose, a $\beta\mu$-reduction suffices. A proof of its confluence is outlined in the second section, using the method of extended parallel reduction.

For formal definitions of the proof-theoretic concepts see chapter 2.5.

5.1 Conversion in $\lambda\mu$PRL and Related Calculi

In the formulation given in chapter 4, the calculus $\lambda\mu$PRL does not contain a cut rule. Since $\lambda\mu$PRL is a top-down refinement calculus, this results in proofs without logical cuts. But considering a cut rule is still sensible since cuts are very useful to structure a proof (corresponding to the use of lemmas). Therefore, in the following we extend $\lambda\mu$PRL by a cut-rule:

$$\begin{array}{l} G, H \vdash C \quad \text{ext } (\lambda x.t)s \\ \quad \text{BY } \texttt{cut}\ i\ T\ x \\ \\ \quad G, H \vdash T \quad \text{ext } s \\ \quad G, x : T, H \vdash C \quad \text{ext } t \end{array}$$

In the presence of cuts, it has to be shown, that every proof containing cuts can be transformed into a proof without cuts. But we will not yet settle on a reduction relation for our calculus. Before, let us reflect on some standard properties we might require.

Usually, the following proof-theoretic properties are of interest:

- strong normalization

- confluence (Church-Rosser-property)
- subformula property of normalized terms
- subject reduction

Proof-theoretic properties form an important area in the research of classical λ-calculi. Accordingly, there exists a number of normalization and confluency results for various λμ-calculi, classical natural deduction systems and also related control calculi (e.g. [Yam04, dG94a, Py98, dG01, Mat01, BHF01, BB93, Par93b, Tat07, NS06, Fuj97, RS94, NT03, NT08, AHS07]).

We will have a look at possible reduction relations for λμPRL alongside these results. However, in regard of the vast amount of different systems, we will do so within a certain predetermined direction.

When considering the general setting, most of it is induced by our reference to the Nuprl system:

- Corresponding to the terms of the Nuprl system, λμPRL terms are untyped. Consequently the reduction relation is defined in an untyped setting, too. This means, that the reduction merely depends on the shape of the derivation, not on additional conditions on the form of the type.
- Nuprl employs a lazy evaluation strategy with β-reduction. The purpose is to ensure termination without restricting the expressiveness of its type system. Again, we would like to stay as close as possible to the system. Therefore we will focus on a β-reduction in a call-by-name-regime for λμPRL, too.

  Nevertheless, due to the classical constructs some additional aspects may need to be regarded.

Our calculus works top-down and the elimination rules operate on the antecedent. As a result, logical cuts can only be introduced into proofs by explicit use of the cut rule. This is different in natural deduction systems: Logical cuts occur when the introduction of a connective is directly followed by its elimination.

However, when we consider computation , such cuts do occur in λμPRL, too. Take the application of a function to some argument of the expected type, i.e. a term of the form $(\lambda x.t\ s)$. This corresponds to a proof where an implication-elimination is followed by its introduction. Now we can evaluate the above term to $t[s/x]$ by the usual β-reduction for λ-terms. Then the proof related to the resulting term may contain further cuts. For λμPRL terms we can identify a number of different cuts. Each kind gives rise to a corresponding reduction relation:

- *logical cuts*: applications of elimination rules followed be introduction rules for the same type (observed in top-down direction) resulting in a non-canonical term with a canonical term of the same type as its principal subterm. This in our notion also includes a rule to reduce terms of the form $[\alpha]\mu\beta.t$. This rule is also referred to as *renaming* (e.g. by Parigot [Par92]). Apart from the μ-renaming, these cuts are redexes of the usual β-reduction in intuitionistic logic. E.g. $(\lambda x.t\ s) \rightsquigarrow_\beta t[s/x]$

- *structural cuts*: if a μ-abstraction is the principal subterm of a non-canonical expression and thus might mask a β-redex. The associated reduction relation is μ-reduction (also referred to as *structural reduction* [Par92] or ζ*-reduction* [PR01]). E.g. $(\mu\alpha.t\ s) \rightsquigarrow_\mu t[[\alpha](u\ s)/[\alpha]u]$
- *permutative cuts*: Since the elimination rules of $\vee$ and $\wedge$ operate on the antecedent, they might mask a redex (see example 5.1 below). We associate two reduction relations depending on the connective δ- and π-reduction for $\vee$ and $\wedge$ respectively. The δ-reductions are also referred to as *permutative conversions* or *commuting conversions*.

The accompanying reduction-relations can be defined as follows for $\lambda\mu$PRL terms. To make the presentation of the rules clearer we will use the following abbreviations:

$$spr(t; x, y.u) \equiv \texttt{let}\ \langle x, y\rangle = t\ \texttt{in}\ u$$
$$dec(t; x.u; y.v) \equiv \texttt{case}\ t\ \texttt{of}\ \texttt{inl}(x) : u \mid \texttt{inr}(y) : v$$

Definition 5.1. *(β-reduction)*

$(\lambda x.u)\ v$	$\rightsquigarrow_\beta$	$u[v/x]$
$spr(\langle v, w\rangle\,; x, y.u)$	$\rightsquigarrow_\beta$	$u[v, w/x, y]$
$dec(inl(w); x.u; y.v)$	$\rightsquigarrow_\beta$	$u[w/x]$
$dec(inr(w); x.u; y.v)$	$\rightsquigarrow_\beta$	$v[w/y]$
$[\alpha]\mu\beta.u$	$\rightsquigarrow_\beta$	$u[\alpha/\beta]$

Definition 5.2. *(μ-reduction)*

$(\mu\alpha.u\ v)$	$\rightsquigarrow_\mu$	$\mu\alpha.u[[\alpha](w\ v)/[\alpha]w]$
$spr(\mu\alpha.z, x, y.u)$	$\rightsquigarrow_\mu$	$\mu\alpha.z[[\alpha]spr(w; x, y.u)/[\alpha]w]$
$dec(\mu\alpha.z; x.u; y.v)$	$\rightsquigarrow_\mu$	$\mu\alpha.z[[\alpha]dec(w; x.u; y.v)/[\alpha]w]$

Definition 5.3. *(δ-reduction)*

$(dec(z; x.u; y.v)\ t)$	$\rightsquigarrow_\delta$	$dec(z; x.(u\ t); y.(v\ t))$
$spr(dec(z; x'.u; y'.v); x, y.t)$	$\rightsquigarrow_\delta$	$dec(z; x'.spr(u; x, y.t); y'.spr(v; x, y.t))$
$dec(dec(z; x'.u'; y'.v'); x.u; y.v)$	$\rightsquigarrow_\delta$	$dec(z; x'.dec(u'; x.u; y.v); y'.dec(v'; x.u; y.v))$

Definition 5.4. *(π-reduction)*

$(spr(z; x, y.u)\ t)$	$\rightsquigarrow_\pi$	$spr(z; x, y.(u\ t))$
$spr(spr(z; x', y'.u'); x, y.u)$	$\rightsquigarrow_\pi$	$spr(z; x', y'.spr(u'; x, y.u)))$
$dec(spr(z; x', y'.u'); x.u; y.v)$	$\rightsquigarrow_\pi$	$spr(z; x', y'.dec(u'; x.u; y.v))$

Apart from reduction rules which concern the different kinds of cuts, works on $\lambda\mu$-calculi contain some further notions of reduction.

- η*-expansions*: transformations like $t \rightsquigarrow_\eta (\lambda x.t\ x)$. These are employed by Pym and Ritter [PR01] to ensure the uniqueness of representation required for their categorial model. To restore confluence in presence of the η-rules they moreover need an additional rule μ_{exp}.
- ν*-reduction*: a combination of η-expansion and μ-reduction (similar to the reduction used by Pym and Ritter above). This is considered in [DP01] in order to formulate an equivalent of Böhm's theorem [Böh68] for $\lambda\mu$ (which is shown to fail).

- *symmetric μ-reduction*: Uses an additional rule μ' for structural reduction. This allows to reduce terms of the form $(s\ \mu\alpha.t)$ where the μ-abstraction is in the argument position. The usual structural reduction merely applies to terms where the μ-abstraction takes the place of the function. Ong and Stewart [OS97] refer to these two kinds of rules as ζ_{arg} and ζ_{fun}.

  The calculus using the symmetric reduction rules is usually denoted by $\lambda\mu\mu'$ (e.g. [Py98, BHF01]). The advantage of this calculus is a stronger notion of normal form: e.g. in the second order variant this means that there are no "false" witnesses for integer values. Instead every integer is represented by a Church numeral as in intuitionistic logic. This links the μ'-rule to the extraction of constructive content from classical proofs. Especially the "output operator" presented by Parigot in [Par93a] is very closely related.

  However, in a strict call-by-name regime μ' is known to destroy the confluence of β-reduction. As remarked in [OS97], this drawback can be fixed by certain restrictions. But these include that the β-reduction is required to call-by-value. Therefore in this work we will be content without symmetric μ-reduction.

- *reduction of control operators*: There is a strong relation between $\lambda\mu$-calculi and λ-calculi with control operators like `call/cc` or Felleisen's $\mathcal{C}$. The reduction of such control operators corresponds to symmetric structural reduction rules. Works on the relations between $\lambda\mu$- and control calculi are e.g. [dG94b, AH08].

  We postpone the relation between $\lambda\mu$PRL and Felleisen's $\mathcal{C}$ to the chapters 6 and 7. It will be considered in context with Murthy's work [Mur90].

The β-, μ- and δ-reduction-relations are the ones usually considered in proofs of strong normalization for classical natural deduction with disjunction [dG01, Mat01, NS06, NT08, DN03]. Pym and Ritter in contrast, use η-expansion rules (following a proof by Ghani [Gha95]) to achieve strong normalization for $\lambda\mu\nu$ and $\lambda\mu\oplus$. However, in [dG01] De Groote remarks that these are not sufficient to ensure the subformula property for normal proofs. To achieve this property in natural deduction the permutative conversions for disjunctions become necessary. However, in $\lambda\mu$PRL the situation which requires the δ- reductions can also arise from the elimination of a conjunction. Consider the following example:

Example 5.1. Let the cut-formula ϕ be a disjunction $A \vee B$ which is not a subformula of the original proposition. But one of the disjuncts is subformula of a conjunction $z : A \wedge C$ in the hypotheses. Therefore ϕ can be proved. Still, for the normal proof to have the subformula property this cut needs to be eliminated.

Let the first refinement step of proving ϕ be by `andE`, resulting in the hypotheses $x_1 : A$ and $y_1 : C$ and the proof term $let\ \langle x_1, y_1\rangle = ...\ in\ ....$ This is followed by an introduction of the disjunction $A \vee B$ by `orI1` $(inl(...))$. The resulting proof goal A can now be proved by the hypothesis rule and the corresponding term is x_1. Accordingly, the proof term of the cut formula is $let\ \langle x_1, y_1\rangle = z\ in\ inl(x_1)$. Now in the proof of the original goal, the cut formula x:$A \vee B$ is eliminated by `orE` $(case\ x\ of\ ...)$. Then in the cut elimination procedure, a redex is masked by the term resulting from the application of `andE`.

An application of the cut-rule results in a term of the form $(\lambda x.t\ u)$ which can be reduced to $t[u/x]$ by β-reduction. But in the example, the term $t[u/x]$ will contain a

subterm $case\ (let\ \langle x_1, y_1\rangle = z\ in\ inl(x_1))\ of\ inl(x_2) : v \mid inr(y_2) : w$. In this term the *let*-construct masks the β-redex which would result from the applications of `orE` and `orI1`. Consequently a permutative conversion for conjunction is required. Employing π-reduction we have

$$\begin{array}{c} case\ (let\ \langle x_1, y_1\rangle = z\ in\ inl(x_1))\ of\ inl(x_2) : v \mid inr(y_2) : w \\ \rightsquigarrow_\pi let\ \langle x_1, y_1\rangle = z\ in\ (case\ inl(x_1)\ of\ inl(x_2) : v \mid inr(y_2) : w) \end{array}$$

Now it is possible to eliminate the logical cut:

$$\begin{array}{c} let\ \langle x_1, y_1\rangle = (case\ inl(s)\ of\ inl(x_2) : v \mid inr(y_2) : w)\ in\ r \\ \rightsquigarrow_\beta let\ \langle x_1, y_1\rangle = v[s/x_2]\ in\ r. \end{array}$$

This also affects the subject reduction property. Without permutative conversions the reduced term might not correspond to a proof in $\lambda\mu$PRL. Still, there would be a corresponding proof in an equivalent bottom-up calculus. But in the top-down direction, the primary subterm of the case- and let-statement can only be a λ-variable. This is an immediate consequence of the left elimination rules.

We conjecture that by the use of $\beta\mu\delta\pi$-reduction it could be shown that

- the typable terms of $\lambda\mu$PRL are strongly normalizing,
- the obtained normal proofs fulfil the subformula property and
- the reduction ensures the preservation of types (subject reduction)

Still, in our current work, we will not further pursue a verification of these properties as it would exceed the scope of this thesis. Currently a confluent call-by-name cut elimination procedure which is similar to Nuprl's operational semantics, but still takes into consideration special role of the μ-operator, suffices. Thus we are content with a $\beta\mu$-reduction-relation. It provides a restricted form of equality reasoning and does not require a deterministic evaluation strategy for confluence. This is important since Nuprl's evaluation strategy is lazy, but non-deterministic.

Therefore section 5.2 concerns a proof of confluence for $\beta\mu$-reduction.

5.2 Confluence of $\beta\mu$-reduction

This section is concerned with the confluence of $\beta\mu$-reduction relation $\twoheadrightarrow_{\beta\mu}$ in $\lambda\mu$PRL. Confluence in $\lambda\mu$-calculi is quite well known. A wide range of confluence results for "pure" $\lambda\mu$ and $\lambda\mu\mu'$ (without sum and products) can e.g. be found in [Py98]. But $\lambda\mu$PRL differs from other $\lambda\mu$-calculi because of its left-hand-side elimination rules. Therefore, we will outline a proof of the confluence of $\beta\mu$-reduction in $\lambda\mu$PRL.

To prove the confluence of a reduction relation, usually one of the following three methods is used:

- Decomposing the reduction relation into several confluent subsystems. If these commute, then by the lemma of Hindley-Rosen [Hin64, Ros73] the original reduction relation is confluent. It is important to note that the confluence of the original relation is not always implied, if merely the confluence of the subsystems can be shown.

- Defining a strongly confluent reduction relation with the same transitive closure as the reduction relation in question. This results in the usual parallel reduction of Martin-Löf and Tait [Bar84]. However, there might exist non-trivial critical pairs which require an extension of the usual method.
- Restricting the considerations to typed terms and showing strong normalization for these. Then it suffices to show local confluence by comparing *critical pairs* (which occur if two different reductions use common symbols). By Newman's lemma [New42] a reduction relation is confluent iff it is locally confluent and strongly normalizing.

Here, we will sketch a proof along the lines of the second method.

We use the following lemma by Barendregt. A strongly confluent reduction relation is said to satisfy the diamond property. The proof of this lemma is given in [Bar84].

Lemma 5.5. *(Barendregt) Let $\rightarrowtail$ be a binary relation on a set and let $\rightarrowtail^*$ be its transitive closure. Then $\rightarrowtail$ satisfying the diamond property implies $\rightarrowtail^*$ satisfying the diamond property.*

The proof of confluence is an extension of Parigot's proof [Par92] which is based on the usual procedure for the λ-calculus due to Martin-Löf and Tait (for a presentation of this method, see [Bar84]). However, there is a gap in the proof outlined by Parigot. This has been independently observed by Py in [Py98] and by Baba, Hirokawa and Fujita in [BHF01].

Obviously, Parigot's one-step-reduction fails to resolve a non-trivial critical pair. Let us consider the term $t \equiv (\mu\alpha.[\alpha]\mu\beta.[\alpha]x\ y)$. This term t contains both a β- and a μ-redex. Thus t can be reduced to $t_1 \equiv (\mu\alpha.[\alpha]x\ y)$ and to $t_2 \equiv \mu\alpha.[\alpha](\mu\beta.[\alpha](x\ y)\ y)$.

Now, t_1 and t_2 would have to be reduced in one step to some common t_3. Otherwise the reduction relation is not strongly confluent. But this is the problem with Parigot's one-step relation: In t_2 the β-redex has to be recovered by an additional step of μ-reduction.

Therefore, it is necessary to extend the rules obtained by the usual parallel reduction method. This is done by the authors in different, but still similar ways. Py adapts a method by Aczel [Acz78] denoted as *generalized parallel reduction*. He extends the usual parallel reduction by two new reduction rules. Baba, Hirokawa and Fujita add just one new rule, which basically subsumes Py's rules.

We will follow the approach by Baba, Hirokawa and Fujita. Due to the elimination rules for conjunction and disjunction, we have to adapt the additional rule respectively. Then the example given above can be resolved: t_1 and t_2 both reduce to $\mu\alpha.[\alpha](x\ y)$ in one step. The necessary one-step reduction relation is defined in the proof of proposition 5.6.

Proposition 5.6. *The reduction relation of $\lambda\mu PRL$ is confluent, i.e. if $u \twoheadrightarrow_{\beta\mu} u_1$ and $u \twoheadrightarrow_{\beta\mu} u_2$, then there exists v such that $u_1 \twoheadrightarrow_{\beta\mu} v$ and $u_2 \twoheadrightarrow_{\beta\mu} v$.*

Proof. We define a new one-step reduction relation $\Rightarrow_1$ with the following properties:

1. $\twoheadrightarrow_{\beta\mu}$ is the transitive closure of $\Rightarrow_1$. This can be shown by set inclusion for relations written as sets of pairs. The idea is that the equality relation induced by $\beta\mu$-reduction-rules is a subset of $\Rightarrow_1$ which is a subset of the transitive closure $\twoheadrightarrow_{\beta\mu}$. Therefore $\twoheadrightarrow_{\beta\mu}$ also is the transitive closure of $\Rightarrow_1$. Details can be found in [Bar84].

2. $\Rightarrow_1$ is strongly confluent.

   It remains to show (2). Then, from (1) and (2) we can deduce that $\twoheadrightarrow_{\beta\mu}$ is confluent by lemma 5.5.

   The reduction relation $\Rightarrow_1$ is inductively defined by:

   (a) $u \Rightarrow_1 u$

   (b) if $u \Rightarrow_1 u'$ then $\lambda x.u \Rightarrow_1 \lambda x.u'$

   (c) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $(u\ v) \Rightarrow_1 (u'\ v')$

   (d) if $u \Rightarrow_1 u'$ then $\mu\alpha.u \Rightarrow_1 \mu\alpha.u'$

   (e) if $u \Rightarrow_1 u'$ then $[\alpha]u \Rightarrow_1 [\alpha]u'$

   (f) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $\langle u, v\rangle \Rightarrow_1 \langle u', v'\rangle$

   (g) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $\texttt{let}\ \langle x, y\rangle = v\ \texttt{in}\ u \Rightarrow_1 \texttt{let}\ \langle x, y\rangle = v'\ \texttt{in}\ u'$

   (h) if $u \Rightarrow_1 u'$ then $\texttt{inl}(u) \Rightarrow_1 \texttt{inl}(u')$

   (i) if $u \Rightarrow_1 u'$ then $\texttt{inr}(u) \Rightarrow_1 \texttt{inr}(u')$

   (j) if $u \Rightarrow_1 u'$, $v \Rightarrow_1 v'$ and $w \Rightarrow_1 w'$ then $\texttt{case}\ w\ \texttt{of}\ \texttt{inl}(x) : u \mid \texttt{inr}(y) : v \Rightarrow_1 \texttt{case}\ w'\ \texttt{of}\ \texttt{inl}(x) : u' \mid \texttt{inr}(y) : v'$

   (k) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $(\lambda x.u\ v) \Rightarrow_1 u'[v'/x]$

   (l) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $(\mu\alpha.u\ v) \Rightarrow_1 \mu\alpha.u'[[\alpha](w\ v')/[\alpha]w]$

   (m) if $u \Rightarrow_1 u'$ then $[\alpha]\mu\beta.u \Rightarrow_1 u'[\alpha/\beta]$

   (n) if $u \Rightarrow_1 u'$, $v \Rightarrow_1 v'$ and $w \Rightarrow_1 w'$ then $\texttt{let}\ \langle x, y\rangle = \langle v, w\rangle\ \texttt{in}\ u \Rightarrow_1 u'[v', w'/x, y]$

   (o) if $u \Rightarrow_1 u'$ and $w \Rightarrow_1 w'$ then $\texttt{case}\ \texttt{inl}(w)\ \texttt{of}\ \texttt{inl}(x) : u \mid \texttt{inr}(y) : v \Rightarrow_1 u'[w'/x]$

   (p) if $u \Rightarrow_1 u'$ and $w \Rightarrow_1 w'$ then $\texttt{case}\ \texttt{inr}(w)\ \texttt{of}\ \texttt{inl}(x) : v \mid \texttt{inr}(y) : u \Rightarrow_1 u'[w'/y]$

   (q) if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $let\ \langle x, y\rangle = \mu\alpha.u\ in\ v \Rightarrow_1 \mu\alpha.u'[[\alpha]let\ \langle x, y\rangle = w\ in\ v'/[\alpha]w]$

   (r) if $u \Rightarrow_1 u'$, $v \Rightarrow_1 v'$ and $w \Rightarrow_1 w'$ then $case\ \mu\alpha.u\ of\ inl(x) : v \mid inr(y) : w \Rightarrow_1 \mu\alpha.u'[[\alpha]case\ z\ of\ inl(x) : v' \mid inr(y) : w'/[\alpha]z]$

   (s) Let $C_i[\_] \in \{(\_\ v_i),\ case\ \_\ of\ inl(x_i) : v_i \mid inr(y_i) : w_i,\ let\ \langle x_i, y_i\rangle = \_\ in\ v_i\}$ and $C'_i[\_] \in \{(\_\ v'_i),\ case\ \_\ of\ inl(x_i) : v'_i \mid inr(y_i) : w'_i,\ let\ \langle x_i, y_i\rangle = \_\ in\ v'_i\}$, $1 \le i \le n$.
   If $u \Rightarrow_1 u'$,$v_1 \Rightarrow_1 v'_1, \ldots, v_n \Rightarrow_1 v'_n$ and $w_1 \Rightarrow_1 w'_1, \ldots, w_n \Rightarrow_1 w'_n$ then $[\beta]C_1[\ldots C_n[\mu\alpha.u]\ldots] \Rightarrow_1 u'[[\beta]C'_1[\ldots C'_n[w]\ldots]/[\alpha]w]$

   From the definition of $\Rightarrow_1$ we obtain 5 different critical pairs:

   i. if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $u[v/x] \Rightarrow_1 u'[v'/x]$

   ii. if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $u[v/x] \Rightarrow_1 u'[[\alpha](wv')/[\alpha]w]$

   iii. if $u \Rightarrow_1 u'$ and $v \Rightarrow_1 v'$ then $u[[\alpha]let\ \langle x, y\rangle = w\ in\ v/[\alpha]w] \Rightarrow_1 u'[[\alpha]let\ \langle x, y\rangle = w\ in\ v'/[\alpha]w]$

   iv. if $u \Rightarrow_1 u'$, $v \Rightarrow_1 v'$ and $w \Rightarrow_1 w'$ then $u[[\alpha]case\ z\ of\ inl(x) : v \mid inr(y) : w/[\alpha]z] \Rightarrow_1 u'[[\alpha]case\ z\ of\ inl(x) : v' \mid inr(y) : w'/[\alpha]z]$

   v. if $u \Rightarrow_1 u'$ then $u[\alpha/\beta] \Rightarrow_1 u'[\alpha/\beta]$

That it is possible to resolve all critical pairs can be shown by induction (for each) on the definition of $\Rightarrow_1$. The base case is $u \Rightarrow u'$ is $u \Rightarrow_1 u'$. This is done by an induction on the form of u. All further cases can be verified by using the induction hypothesis. The only interesting cases are related to the gap in Parigot's proof and arise from items (2(s)ii),(2(s)iii) and (2(s)iv). The proof for item (2(s)ii) and rule (2m) can basically be found in [BHF01]. The cases for (2(s)iii) and (2(s)iv) can be shown analogously. Therefore we will be content with an example for (2(s)iii) and rule (2s) to illustrate the procedure for this rule. Then the corresponding cases for (2(s)ii) and (2(s)iv) could again be done analogously.

We will now proceed as follows: Let us first take a look at the base case and one simple induction step for (2j) on item (2(s)iii). Then we will go on to one of the cases concerning structural reduction. As exemplary case we show the proof for (2(s)iii) and the rule (2s). For space reasons, the *case*- and *let*-constructs are once more abbreviated by *decide* and *spread*.

- base case: $u \Rightarrow_1 u'$ is $u \Rightarrow_1 u$. Then we have to show by induction on the structure of u that $u[[\alpha]let\ \langle x,y\rangle = s\ in\ t/[\alpha]s] \Rightarrow_1 u[[\alpha]let\ \langle x,y\rangle = s\ in\ t'/[\alpha]s]$. This can be done in a table:

u	lhs	rhs	comment
base cases:			
x	x	x	OK
$[\alpha]v,\ \alpha \notin FV(v)$	$[\alpha]let\ \langle x,y\rangle = w\ in\ v$	$[\alpha]let\ \langle x,y\rangle = w\ in\ v'$	OK

induction hypothesis (IH1): $r[[\alpha]let\ \langle x,y\rangle = s\ in\ t/[\alpha]s] \Rightarrow_1 r[[\alpha]let\ \langle x,y\rangle = s\ in\ t'/[\alpha]s]$, for short $r[\,] \Rightarrow_1 r[\,']$
induction step: $r \to E[r]$ for all contexts $E[\_]$

$\lambda x.v$	$\lambda x.v[\,]$	$\lambda x.v[\,']$	by IH1
$(v\ w)$	$(v[\,]\ w[\,])$	$(v[\,']\ w[\,'])$	— ‖ —
$\mu\alpha.v$	$\mu\alpha.v[\,]$	$\mu\alpha.v[\,']$	— ‖ —
$[\alpha]v,\ \alpha \in FV(v)$	$[\alpha]let\ \langle x,y\rangle = v[\,]\ in\ t$	$[\alpha]let\ \langle x,y\rangle = v[\,']\ in\ t'$	— ‖ —
$\langle v,w\rangle$	$\langle v[\,],w[\,]\rangle$	$\langle v[\,'],w[\,']\rangle$	— ‖ —
$let\ \langle x,y\rangle = z\ in\ v$	$let\ \langle x,y\rangle = z[\,]\ in\ v[\,]$	$let\ \langle x,y\rangle = z[\,']\ in\ v[\,']$	— ‖ —
$inl(v)$	$inl(v[\,])$	$inl(v[\,'])$	— ‖ —
$inr(v)$	$inr(v[\,])$	$inr(v[\,'])$	— ‖ —
$decide(z;x.v;y.w)$	$decide(z[\,];x.v[\,];y.w[\,])$	$decide(z[\,'];x.v[\,'],y.w[\,'])$	— ‖ —

- induction hypothesis(IH2): $r[[\alpha]let\ \langle x,y\rangle = s\ in\ t/[\alpha]s] \Rightarrow_1 r'[[\alpha]let\ \langle x,y\rangle = s\ in\ t'/[\alpha]s]$
- induction step: $r \to E[r]$
- exemplary case for common parallel reduction cases: rule (2j).
  $u \Rightarrow_1 u'$ is $decide(z;x.v;y.w) \Rightarrow_1 decide(z';x.v';y.w')$ and a direct consequence

of $z \Rightarrow_1 z'$, $v \Rightarrow_1 v'$ and $w \Rightarrow_1 w'$. Let $\sigma \equiv [[\alpha]let\ \langle x, y\rangle = s\ in\ t/[\alpha]s]$ and $\sigma' \equiv [[\alpha]let\ \langle x, y\rangle = s\ in\ t'/[\alpha]s]$. Then

$$\begin{aligned} u\sigma &\equiv decide(z\sigma; x.v\sigma; y.w\sigma) \\ &\Rightarrow_1 decide(z\sigma'; x.v\sigma'; y.w\sigma') \quad \text{,by IH2,} \\ &\equiv u'\sigma'. \end{aligned}$$

- exemplary case for structural rules: rule (2s).
  Let $C_i[\_] \in \{(\_\ v_i),\ decide(\_; x_i.v_i; y_i.w_i),\ spread(\_; x_i, y_i.v_i)\}$ and
  $C'_i[\_] \in \{(\_\ v'_i),\ decide(\_; x'_i.v'_i; y'_i.w'_i),\ spread(\_; x'_i, y'_i.v'_i)\}$, $1 \le i \le n$.

  Now $u \Rightarrow_1 u'$ is of the form
  $[\beta]C_1[\dots C_n[\mu\gamma.u_1]\dots] \Rightarrow_1 u'_1[[\beta]C'_1[\dots C'_n[w]\dots]/[\gamma]w]$
  and we know the induction hypothesis is valid for $u_1 \Rightarrow_1 u'_1$,$v_1 \Rightarrow_1 v'_1, \dots, v_n \Rightarrow_1 v'_n$
  and $w_1 \Rightarrow_1 w'_1, \dots, w_n \Rightarrow_1 w'_n$.

  Consequently $C_i[\_][[\alpha]spread(w; x, y.v)/[\alpha]w] \Rightarrow_1 C'_i[\_][[\alpha]spread(w; x, y.v')/[\alpha]w]$.
  For space reasons let $C_i\sigma[\_] \equiv C_i[\_][[\alpha]spread(w; x, y.v)/[\alpha]w]$ and $C'_i\sigma[\_] \equiv C'_i[\_][[\alpha]spread(w; x, y.v')/[\alpha]w$

  Two cases arise, depending on the μ-variables: $\alpha = \beta$ and $\alpha \neq \beta$.

  - $\alpha = \beta$. Then

    $$\begin{aligned} &u[[\alpha]spread(w; x, y.v)/[\alpha]w] \\ &= [\alpha]C_1[\dots C_n[\mu\gamma.u_1]\dots][[\alpha]spread(w; x, y.v)/[\alpha]w] \\ &= [\alpha]spread(C_1\sigma[\dots C_n\sigma[\mu\gamma.u_1[[\alpha]spread(w; x, y.v)/[\alpha]w]]\dots]; x, y.v) \\ &=: u_2, \end{aligned}$$

    $$\begin{aligned} &u'[[\alpha]spread(w; x, y.v')/[\alpha]w] \\ &= u'_1[[\alpha]C'_1[\dots C'_n[w]\dots]/[\gamma]w][[\alpha]spread(w; x, y.v')/[\alpha]w] \\ &= u'_1[[\alpha]spread(C'_1\sigma[\dots C'_n\sigma[w]\dots]; x, y.v')/[\gamma]w] \\ &=: u'_2. \end{aligned}$$

    Using the induction hypothesis for $u_1 \Rightarrow_1 u'_1$, we know
    $u_1[[\alpha]spread(w; x, y.v)/[\alpha]w] \Rightarrow_1 u'_1[[\alpha]spread(w; x, y.v')/[\alpha]w]$
    Therefore by rule (2s) $u_2 \Rightarrow_1 u'_2$.
  - $\alpha \neq \beta$. Then

    $$\begin{aligned} &u[[\alpha]spread(w; x, y.v)/[\alpha]w] \\ &= [\beta]C_1[\dots C_n[\mu\gamma.u_1]\dots][[\alpha]spread(w; x, y.v)/[\alpha]w] \\ &= [\beta]C_1\sigma[\dots C_n\sigma[\mu\gamma.u_1[[\alpha]spread(w; x, y.v)/[\alpha]w]]\dots] \\ &=: u_2, \end{aligned}$$

    $$\begin{aligned} &u'[[\alpha]spread(w; x, y.v')/[\alpha]w] \\ &= u'_1[[\beta]C'_1[\dots C'_n[w]\dots]/[\gamma]w][[\alpha]spread(w; x, y.v')/[\alpha]w] \\ &= u'_1[[\beta]C'_1\sigma[\dots C'_n\sigma[w]\dots]; /[\gamma]w] \end{aligned}$$

$=: u_2'$.

Using the induction hypothesis for $u_1 \Rightarrow_1 u_1'$, we know
$u_1[[\alpha]spread(w; x, y.v)/[\alpha]w] \Rightarrow_1 u_1'[[\alpha]spread(w; x, y.v')/[\alpha]w]$
Therefore by rule (2s) $u_2 \Rightarrow_1 u_2'$.

- The other cases can be verified analogously: The "common" cases like the proof for 2(s)iii and rule (2j); the "structural" cases similar to 2(s)iii and rule (2s).

= Having resolved the critical pairs, it can easily be shown by induction on the definition of $u \Rightarrow_1 u'$ that for all $u \Rightarrow_1 u''$ there is an u''' such that $u' \Rightarrow_1 u'''$ and $u'' \Rightarrow_1 u'''$. Thus $\Rightarrow_1$ is confluent, and by lemma 5.5 $\twoheadrightarrow_{\beta\mu}$ is confluent as well.

□

Chapter 6

On Classical and Intuitionistic Terms of $\lambda\mu$PRL

This chapter is concerned with evaluation of a different kind. Although being related to normalization in the sense of unique data representation, it focuses on another purpose. Our motivation for a further evaluation of normal proofs (in the sense of the reduction relation presented in the previous chapter), is to decide whether a proof relies on a classical or a constructive argument. Such a procedure would enable a proof system to distinguish classical from constructive proofs on a meta-level.

The first step in the development of this evaluation procedure is to consider in which cases a proof in $\lambda\mu$PRL is non-constructive and how this is reflected in the respective proof term.

It is easy to see, that an intuitionistic fragment of $\lambda\mu$PRL can be obtained by taking the calculus minus the rules `abort1` and `abort2`. But simply classifying proofs which use these rules as non-constructive does also exclude proofs which are constructively valid. After all, the conclusion might have been exchanged during the proof, albeit completely irrelevant for the completion of the proof. Consequently we need a more detailed analysis of the proof structure.

If we have a look at the proof tree of the derivation of a sentence in $\lambda\mu$PRL, the root is the sentence itself, and every leaf of the proof tree must contain an instance of the rule `hypothesis`. Accordingly, the constructivity of the proof depends on how the hypotheses used in the application of `hypothesis` have been obtained.

In $\lambda\mu$PRL, the only rule which moves formulae from the succedent to the antecedent is the introduction of an implication, which is represented in the proof term by a λ-abstraction. But the crucial point is the influence of context formulae in the proof tree. In the intuitionistic fragment of $\lambda\mu$PRL (as described above) two rules "destroy" the formula in the succedent: the elimination of an implication and the introduction of a disjunction. This is a usual property of these rules in single-conclusioned intuitionistic logic.

But in the presence of the `abort`-rules, the formulae which are "lost" in the intuitionistic case can be saved by moving them to the context before one of these "destructive" rules is applied, and recovering them later on. Obviously, if the μ-variable of the recovered formula is α, the application of `abort` would result in a subterm of the form $[\alpha]t$. Now, the following cases arise which would render the proof classical:

1. the subtree rooted in an introduction of disjunction contains a leaf that proves one

disjunct (or one of its subformulae) by a hypothesis obtained from the other disjunct.

2. the subtree in which the premiss of an eliminated implication is to be proven does not prove the premiss itself but a formula from the context, depending on a hypothesis obtained from the premiss (if the context formula was proven without an additional hypothesis this would have shown the unnecessity of using the eliminated implication in this place and consequently would have been removed within the process of cut elimination.)

Let us take a look at the proof terms resulting from proofs to which these two cases apply.

1. In this case the proof term will contain a subterm

$$\mu\alpha.\ \ldots inl(\ldots \lambda x.\ \ldots [\alpha] \ldots inr(\ldots x))$$
$$\text{or}$$
$$\mu\alpha.\ \ldots inr(\ldots \lambda x.\ \ldots [\alpha] \ldots inl(\ldots x))$$

   An obvious example (and in fact the simplest instance) of this case is the proof of the law of the excluded middle $A \vee (A \supset \bot)$. E.g., it is represented by the term $\mu\alpha.[\alpha]inr(\lambda x.[\alpha]inl(x))$

2. This scenario will result in a subterm

$$\mu\alpha.\ \ldots (t\ (\ldots \lambda y.\ \ldots [\alpha] \ldots y))$$

   where t is the respective term which represents the eliminated implication the premiss of which is proven by a switch of the goal formula. Example are the classical axiom $\neg\neg A \supset A$ and Peirce's Law $(((P \supset Q) \subset P) \supset P)$. These can be represented by the terms $\lambda x.\mu\alpha.(x\ \lambda y.[\alpha]y)$ and $\lambda f.\mu\alpha.[\alpha](f\ \lambda y.\mu\beta.[\alpha]y)$

As one can see, both cases involve instances of the prominent classical theorems that do not hold intuitionistically. This reflects once more on the well-known fact that a classical calculus can be obtained from an intuitionistic one by adding one of these as axioms.

Now the question is, how to generally identify such structures in proof terms. One possibility seems to be an adaption of a method by Parigot [Par93a]. Since in his second order $\lambda\mu$-calculus normalized proofs of natural numbers are not only represented by Church numerals but also by infinitely many "false" witnesses, he proposes the use of an output operator. This operator is related to the additional reduction rule of the symmetric $\lambda\mu$-calculus mentioned in the previous chapter. In [Par93a] Parigot shows that there is an output operator which can compute the constructive term (i.e. the Church numeral) corresponding to a natural number from every false witness.

It might be possible to use a similar concept to return the constructive term, if a term of $\lambda\mu$PRL relies on a constructive argument and "something else" otherwise. However, using this output operator is very similar to interpreting the μ-operator as control-operator and reducing it like Felleisen's operator $\mathcal{C}$. Since this is the evaluation used by Murthy in the context of Nuprl's logic, it might be more reasonable to consider adapting this method for our purpose.

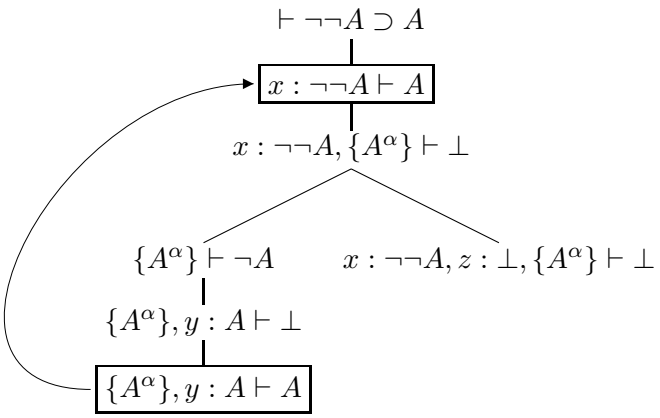
6.1 Murthy's Evaluation

Let us now take a look at two evaluation rules which interpret the μ-abstraction as control operator and an accompanying evaluation strategy. The rules correspond to the reduction relation of Felleisen's calculus $\lambda_{\mathcal{C}}$ [FFKD87, FH92] and have also been used by Murthy [Mur91b] in his deterministic evaluation of Σ_1^0 and Π_2^0 sentences in Peano Arithmetic. Murthy's evaluation strategy is a deterministic head-reduction strategy. The μ-operator plays the role of Felleisen's control operator $\mathcal{C}$ and we introduce a new unary operator $\mathcal{A}$ which does not bind any variables, although in presence of the reduction rules below it could also be expressed by a μ-abstraction with a dummy variable not occurring in the subterm: $\mu\alpha^*.t, \alpha^* \notin FV(t)$.

Definition 6.1. *(ϵ-reduction)*

$$\begin{array}{lll} E[\mu\alpha.t] & \rightsquigarrow_\epsilon & t[\mathcal{A}(E[w])/[\alpha]w] \\ E[\mathcal{A}(t)] & \rightsquigarrow_\epsilon & t \end{array}$$

If we now return to our examples for propositions which do not hold intuitionistically, it is interesting to observe what happens to the proof terms if we apply the rules of ϵ-reduction. To illustrate the matter, we first take a look at the proof tree of the double negation elimination rule represented by the term $\lambda x.\mu\alpha.(x\ \lambda y.[\alpha]y)$.



Applying the ϵ-reduction rules to the proof term now basically means to replace the subtree rooted in the first box by the subtree rooted in the second box (which however is a leaf in the example):

$$\lambda x.\mu\alpha.(x\ \lambda y.[\alpha]y) \rightsquigarrow_\epsilon (x\ \lambda y.\mathcal{A}(\lambda x.y)) \rightsquigarrow_\epsilon \lambda x.y$$

Obviously, this reduction results in a non-closed term. This problem does not occur in Murthy's system, since it is a first order calculus for arithmetic in which the only atomic type is $\mathbb{N}$ and it furthermore is restricted to Σ_1^0 sentences (where classical logic is conservative over intuitionistic logic).

Of course for non-decidable propositions such a reduction is not an equality transformation on proofs, but the resulting free variable does signal the non-constructivity of the proof. Since its binding λ-abstraction has occurred in between the μ-abstraction and the re-activation of the associated context formula, we can conclude, that this variable is not accessible in an intuitionistic proof. To illustrate this point, we could also regard

the different μ-contexts as subroutines, so that in the intuitionistic case each subroutine merely has access to its own namespace while in the classical case all variables are globally accessible.

To avoid a non-closed term we could alter the reduction rules by substituting either a new constant $any_{x_1}, \ldots, any_{x_n}$ or simply one special constant any for every free variable $x_1, \ldots, x_n$ (if we are simply interested in whether a classical argument has been used or not the latter is sufficient):

$$\begin{array}{l}\text{Variant 1: } E[\mathcal{A}(t)] \rightsquigarrow_{\epsilon_1} t[FV(t)_i/any_i] \text{ or} \\ \text{Variant 2: } E[\mathcal{A}(t)] \rightsquigarrow_{\epsilon_2} t[FV(t)/any]\end{array}$$

Still, we require another prerequisite to be able to tell apart non-constructive from constructive proofs. This depends on the order in which the rules of the proof are applied. The problem is, that not only λ-abstractions bind variables, but also the terms associated with $\vee$- and $\wedge$-elimination. Therefore we could ,,loose" variables since in a calculus in which elimination rules operate on the antecedent it is possible to apply these elimination rules independently from the form of the succedent – e.g. in between applications of the rules `mu` and `abort`.

However, their independence of the type of the succedent also contains the chance to move the application of the respective rule in the proof tree. The "secure" position for such a conjunction or disjunction elimination is directly below the function introduction which has added said conjunction or disjunction to the hypotheses (the merely have to avoid variable capture). Consider the following simple example which obviously has both a classical and an intuitionistic proof:

Example 6.1. For space reasons the extract terms are omitted in the proof trees and can be found below.

$$\begin{array}{l}H \equiv z : A \wedge B, \{\{(A \vee \neg A)^\alpha\}\} \\ H' \equiv x_1 : A, y : B, \{\{(A \vee \neg A)^\alpha\}\}, x_2 : A \\ H'' \equiv z : A \wedge B, \{\{(A \vee \neg A)^\alpha\}\}, x_2 : A \\ H''' \equiv x_1 : A, y : B, \{\{(A \vee \neg A)^\alpha\}\}, x_2 : A\end{array}$$

We take a look at three possible derivations of the proposition $(A \wedge B) \supset (A \vee \neg A)$ in $\lambda\mu$PRL:

$\vdash (A \wedge B) \supset (A \vee \neg A)$ `ext` t_1
by lambdaI z
|_ $z : A \wedge B \vdash A \vee \neg A$
by mu1 α
|_ $z : A \wedge B, \{\{(A \vee \neg A)^\alpha\}\} \vdash \bot$
by abort2 α
|_ $H \vdash A \vee \neg A$
by orI1
|_ $H \vdash \neg A$
by lambdaI x_2
|_ $H'' \vdash \bot$
by abort2 α
|_ $H'' \vee \neg A$
by orI2
|_ $H'' \vdash A$
by andE
|_ $H''' \vdash A$
by hypothesis 1

$\vdash (A \wedge B) \supset (A \vee \neg A)$ `ext` t_2
by lambdaI z
|_ $z : A \wedge B \vdash A \vee \neg A$
by mu1 α
|_ $z : A \wedge B, \{\{(A \vee \neg A)^\alpha\}\} \vdash \bot$
by abort2 α
|_ $H \vdash A \vee \neg A$
by orI1
|_ $H \vdash \neg A$
by lambdaI x_2
|_ $H'' \vdash \bot$
by abort2 α
|_ $H'' \vee \neg A$
by orI2
|_ $H'' \vdash A$
by andE
|_ $H''' \vdash A$
by hypothesis 4

$\vdash (A \wedge B) \supset (A \vee \neg A)$ `ext` t_3
by lambdaI z
|_ $z : A \wedge B \vdash A \vee \neg A$
by mu1 α
|_ $z : A \wedge B, \{\{(A \vee \neg A)^\alpha\}\} \vdash \bot$
by andE
|_ $H' \vdash A$
by abort2 α
|_ $H' \vdash A \vee \neg A$
by orI1
|_ $H' \vdash \neg A$
by lambdaI x'
|_ $H''' \vdash \bot$
by abort2 α
|_ $H''' \vdash A \vee \neg A$
by orI2
|_ $H''' \vdash A$
by hypothesis 1

The three derivations result in the following proof terms:

$$t_1 \equiv \lambda z.\mu\alpha.[\alpha]inr(\lambda x_2.[\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_1))$$
$$t_2 \equiv \lambda z.\mu\alpha.[\alpha]inr(\lambda x_2.[\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_2))$$
$$t_3 \equiv \lambda z.\mu\alpha.let\ \langle x_1, y\rangle = z\ in\ [\alpha]inr(\lambda x_2.[\alpha]inl(x_1))$$

If we applied the ϵ-reduction to these terms without any previous conversion, we would get (where the boxes mark the freed variables):

$$t_1 \rightsquigarrow_\epsilon^* \lambda z.inl(let\ \langle x_1, y\rangle = z\ in\ x_1)$$
$$t_2 \rightsquigarrow_\epsilon^* \lambda z.inl(let\ \langle x_1, y\rangle = z\ in\ \boxed{x_2})$$
$$t_3 \rightsquigarrow_\epsilon^* \lambda z.inl(\boxed{x_1})$$

Now we have three different cases:

1. t_1 reduces to the equivalent constructive proof term
2. t_2 reduces to a term containing a free variable which is as expected since it relies on a non-constructive argument and proves the law of the excluded middle without using the premiss $A \wedge B$
3. t_3 also reduces to a non-closed term, however it does not rely on a classical argument, but the term representing the elimination of the conjunction is simply lost in the reduction.

So, while the ϵ-reduction of t_1 and t_2 gives the desired results, the reduction of t_3 does not. For this reason it would be necessary to permute the proof term with respect to the position of the $\vee$- and $\wedge$-eliminations before applying a further evaluation.

And still, there is something else to pay attention to: Classical reasoning has an intrinsic notion of non-determinism, which also affects term languages with control operators. Usually this non-determinism is either restricted by the reduction rules, a deterministic evaluation strategy or the term language is accepted as being non-deterministic and non-confluent [UB01] (a discussion can be found in the previous chapter).

We can follow Murthy by fixing a deterministic evaluation strategy, but if we are interested in not "loosing" constructive proofs by the choice of the evaluation strategy, we need to evaluate the term under more than one strategy and then have a look at the resulting terms whether they contain occurrences of free variables (or *any*-constants).

So, the ϵ reduction presented above is well suited to get an idea of problem. However, due to the difficulties in presence of left-hand side elimination rules and the non-determinism of classical proofs, the question arises, whether there is a more straightforward procedure to determine whether a proof relies on a classical argument or not. As we have seen above, reducing the proof term by the ϵ-reduction rules results in free λ-variables, either if the proof is classical or if disjunctions or conjunctions have been eliminated in an inopportune place. Now we would like to be able to differ between latter and former, not only between free variables or bound variables. However, this cannot be accomplished once the term has been reduced by ϵ-reduction. We would either need perform another permutation of the proof in advance moving applications of elimination rules - or find another method.

6.2 Distinguishing Terms by Crolard's μ-safety

Such another method is presented by Crolard in [Cro02]. He presents a constructive restriction $\overline{CND}^{r}_{\vee}$ of Parigot's $\lambda\mu$ (extended by disjunction and a cut rule) annotating the inference rules so that a set of interdependencies between hypotheses and conclusions is defined. The intuition is to keep the multi-conclusion, and the opportunity to exchange the active formula, but to restrict the implication introduction in a way, that the interdependencies make sure that only the hypotheses linked to the active conclusion can be used. This idea is very similar to the constructive multi-conclusioned logic FIL ("Full Intuitionistic Logic") by Pereira and de Paiva [PdP05].

Crolard furthermore defines an extension of classical $\lambda\mu$ by disjunction and cut as $\lambda\mu+$. To determine whether a $\lambda\mu+$-term t can be typed in $\overline{CND}^{r}_{\vee}$ he introduces the notion of the *scope* $S_\delta(t)$ of a μ-variable δ and defines *μ-safe* terms as those $\lambda\mu+$-terms which represent constructive proofs. The scopes of the μ-variables are exactly the interdependencies of $\overline{CND}^{r}_{\vee}$.

Definition 6.2. *(Crolard [Cro02]) A $\lambda\mu+$-term t is safe with respect to μ-contexts (μ-safe for short) iff for any subterm of t which has the form $\lambda x.u$, for any free μ-variable δ of u, $x \notin S_\delta(u)$.*

We can proceed in the same way for $\lambda\mu$PRL – basically, if a term is not μ-safe, the λ-variables contained in the scope of the respective μ-variables are exactly the ones which would occur freely after ϵ-reduction, if the corresponding proof was non-constructive. The variables bound by the *let*- or *case*-constructs do not interfere, since this problem is solved by the construction of the set S_δ. Therefore, we will proceed by giving a constructive

restriction of λμPRL with annotations and defining the scope of a μ-variable for λμPRL. We adopt Crolard's notation and adapt the following abbreviation:

$$U, V sets. \qquad U[V/W] = \begin{cases} U \setminus W \cup V \ if \ W \cap U \neq \emptyset \\ U \ otherwise \end{cases}$$

Definition 6.3. *(Scope of a μ-variable) By induction on a term t of λμPRL we define the set $S_{[]}(t)$ of the free λ-variables which occur* out of the scope of any *μ-variable in t and the set $S_\delta(t)$ of the free λ-variables which occur* in *the scope of a free μ-variable δ in t.*

- $S_{[]}(x) = x$
  $S_\delta(x) = \emptyset$

- $S_{[]}(\lambda x.u) = S_{[]}(u) \setminus \{x\}$
  $S_\delta(\lambda x.u) = S_\delta(u) \setminus x$

- $S_{[]}(u\ v) = S_{[]}(u) \cup S_{[]}(v)$
  $S_\delta(u\ v) = S_\delta(u) \cup S_\delta(v)$

- $S_{[]}([\alpha]u) = \emptyset$
  $S_\delta([\alpha]u) = S_\delta(u)\ for\ any\ \delta \neq \alpha$
  $S_\alpha([\alpha]u) = S_\alpha(u) \cup S_{[]}(u)$

- $S_{[]}(\mu\alpha.u) = S_\alpha(u)$
  $S_\delta(\mu\alpha.u) = S_\delta(u)$

- $S_{[]}(\langle u, v\rangle) = S_{[]}(u) \cup S_{[]}(v)$
  $S_\delta(\langle u, v\rangle) = S_\delta(u) \cup S_\delta(v)$

- $S_{[]}(let\ \langle x, y\rangle = z\ in\ u) = S_{[]}(u)[S_{[]}(z)/\{x, y\}]$
  $S_\delta(let\ \langle x, y\rangle = z\ in\ u) = S_\delta(u)[S_{[]}(z)/\{x, y\}] \cup S_\delta(z)$

- $S_{[]}(inl(u)) = S_{[]}(u)$
  $S_\delta(inl(u)) = S_\delta(u)$
  $S_{[]}(inr(u)) = S_{[]}(u)$
  $S_\delta(inr(u)) = S_\delta(u)$

- $S_{[]}(case\ z\ of\ inl(x) : u \mid inr(y) : v) = S_{[]}(u)[S_{[]}(z)/\{x\}] \cup S_{[]}(v)[S_{[]}(z)/\{y\}]$
  $S_\delta(case\ z\ of\ inl(x) : u \mid inr(y) : v) = S_\delta(u)[S_{[]}(z)/\{x\}] \cup S_\delta(v)[S_{[]}(z)/\{y\}] \cup S_\delta(z)$

Remark The above definition of the scope requires that cuts have been eliminated from the term t in advance, since we cannot distinguish whether a λ-abstraction results from a cut or from the introduction of an implication. This is not a serious problem: if we evaluate a term to check its constructivity this can be combined with a reduction. However we could also deal with terms containing cuts, but we would have to identify another term with the application of the cut-rule:

$$\frac{G, H \vdash C \quad \text{ext } let\ x = v\ in\ u}{\text{BY } \texttt{cut}\ i\ T\ x}$$

$$\frac{G, H \vdash T \quad \text{ext } v}{G, x : T, H \vdash C \quad \text{ext } u}$$

Then definition 6.3 could be extended by:

- $S_{[]}(let\ x = v\ in\ u) = S_{[]}(u)[S_{[]}(v)/x]$
  $S_\delta(let\ x = v\ in\ u) = S_\delta(u)[S_{[]}(v)/x] \cup S_\delta(v)$

As a consequence the inductive definition of a scope could also be used to distinguish between constructive and non-constructive proof terms in presence of a cut. Still, the cut elimination procedure would need to be adapted to resolve the new construct. This could be achieved by adding the new reduction rule:

$$let\ x = v\ in\ u \rightsquigarrow_\beta u[v/x]$$

It can be verified that this extension of the reduction relation would not break the confluence.

Now we can define μ-safety for terms of $\lambda\mu$PRL analogously to definition 6.2.

Definition 6.4. *(μ-safety of $\lambda\mu$PRL-terms) A term t of $\lambda\mu$PRL is called safe in respect to μ-contexts, or μ-safe for short, iff for any subterm of t of the form $\lambda x.u$, $x \notin S_\delta(u)$ for any free μ-variable δ.*

Example 6.2. We return to the terms of example 6.1. It is easy to see that the notion of μ-safety provides the desired means to distinguish intuitionistically valid from non-constructive terms:

- $t_1 \equiv \lambda z.\mu\alpha.[\alpha]inr(\lambda x_2.[\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_1))$: We have to consider the subterms of the λ-abstractions λz and λx_2. Then t_1 is μ-safe, since z is declared out of the scope of any μ-variable and so may occur in anywhere in the subterm of its abstraction and x_2 does not occur in the scope of α which is the only free μ-variable in the subterm of its abstraction: $x_2 \notin S_\alpha([\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_1))(= \{z\})$ and there is no free μ-variable in $\mu\alpha.[\alpha]inr(\lambda x_2.[\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_1))$.

- $t_2 \equiv \lambda z.\mu\alpha.[\alpha]inr(\lambda x_2.[\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_2))$: t_2 is not μ-safe as x_2 is contained in the scope of α: $S_\alpha([\alpha]inl(let\ \langle x_1, y\rangle = z\ in\ x_2)) = \{x_2\}$.

- $t_3 \equiv \lambda z.\mu\alpha.let\ \langle x_1, y\rangle = z\ in\ [\alpha]inr(\lambda x_2.[\alpha]inl(x_1))$: t_3 is μ-safe basically for the same reason as t_1. $x_2 \notin S_\alpha([\alpha]inl(x_1))(= \{z\})$ and there is no free μ-variable in $\mu\alpha.let\ \langle x_1, y\rangle = z\ in\ [\alpha]inr(\lambda x_2.[\alpha]inl(x_1))$.

If we have a look at our first analysis of non-constructive $\lambda\mu$PRL-terms, it is also easy to see that in both of the identified cases these are not μ-safe.

To verify, that μ-safe terms are indeed the ones which correspond to constructive proofs, we now formulate a constructive restriction of $\lambda\mu$PRL which we will refer to as $\lambda\mu_r$PRL. As a starting point we again use an idea employed in [Cro02]:

Crolard formulates a calculus $\overline{CND}_\vee$ by annotating the formulae of the conclusion by their interdependencies with λ-variables. These sets of interdependencies are exactly the sets from definition 6.3, i.e. $S_{[]}$ corresponds to the interdependencies of the active conclusion and the interdependencies of each μ-variable δ naming a passive formula are contained in S_δ. He then restricts this calculus to the constructive version $\overline{CND^r_\vee}$. To ensure constructivity, he only allows the introduction of an implication if the λ-variable of the abstraction does not occur in any of the scopes (sets of interdependencies) of the passive formula. The reason why this method renders the calculus constructive is that it keeps the implication local as the respective hypothesis cannot be used in the proof of any of the passive formulae – otherwise the variable would occur in the scope of the μ-variable associated with the respective formula.

Still, because our calculus works in top-down direction, we need some more notation than in a bottom-up calculus, since the inductive definition of the scope (and equivalently the interdependencies) means that the scopes of the μ-variables can like the proof term only be computed when the proof is completed. As a consequence, at the point where the implication introduction is applied, there is not enough information on the scopes available to restrict the rule as Crolard does. However, we can implement this restriction by the use of some more annotation. We simply have to add a sets of "forbidden" λ-variables and alter the `hypothesis` rule such that these forbidden hypotheses may not be used. The construction of these sets of forbidden variables can loosely be thought of as a recursive check of μ-safety by stating for every term which λ-variables may not occur freely in its subterms.

The sequents of $\lambda\mu_r$PRL are sequents of $\lambda\mu$PRL with the following extensions: The μ-variables and the conclusion are each annotated by a tuple of sets $\mathcal{S}|\mathcal{F}$ where $\mathcal{S}$ is a placeholder for the interdependencies and $\mathcal{F}$ contains the forbidden λ-variables. The interdependencies of μ-variables and conclusion with λ-variables are computed from the completed proof as is the proof term. The construction of the set of interdependencies in correspondence to the rules applied obviously reflects and is consistent with definition 6.3. This draws the connection between the calculus and the concept of μ-safety.

The rules `lambdaI`, `functionE`, `andI`, `andE` and `orE` require to express operations being performed on the annotation of each μ-variable. Let

$$\mathcal{S}_{\mu H} = \{\mathcal{S}_\delta \mid \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H\}$$

$$\mathcal{F}_{\mu H} = \{\mathcal{F}_\delta \mid \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H\}$$

$$H(\mathcal{S}1_{\mu G}|\mathcal{F}1_\mu G) = \forall \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H, \mathcal{S}1_\delta \in \mathcal{S}_{\mu G}, \mathcal{F}1_\delta \in \mathcal{F}_{\mu G}. \\ (\mathcal{S}_\delta = \mathcal{S}1_\delta \wedge \mathcal{F}_\delta = \mathcal{F}1_\delta)$$

$$H(\mathcal{S}1_{\mu G} \cup \mathcal{S}2_{\mu G}|\mathcal{F}_\mu) = \forall \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H, \mathcal{S}1_\delta \in \mathcal{S}1_{\mu G}, \mathcal{S}2_\delta \in \mathcal{S}2_{\mu G}, \mathcal{F}_\delta \in \mathcal{F}_{\mu G}. \\ (\mathcal{S}_\delta = \mathcal{S}1_\delta \cup \mathcal{S}2_\delta \wedge \mathcal{F}_\delta = \mathcal{F}1_\delta)$$

$$Uset. \quad H(\mathcal{S}1_{\mu G}|\mathcal{F}1_\mu \cup U) = \forall \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H, \mathcal{S}1_\delta \in \mathcal{S}1_{\mu G}, \mathcal{F}1_\delta \in \mathcal{F}1_{\mu G}.$$

$$(\mathcal{S}_\delta = \mathcal{S}1_\delta \wedge \mathcal{F}_\delta = \mathcal{F}1_\delta \cup U)$$

$$U, V sets. \quad H(\mathcal{S}1_{\mu G}[U/V]|\mathcal{F}1_\mu) = \forall \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H, \mathcal{S}1_\delta \in \mathcal{S}1_{\mu G}, \mathcal{F}1_\delta \in \mathcal{F}1_{\mu G}. (\mathcal{S}_\delta = \mathcal{S}1_\delta[U/V] \wedge \mathcal{F}_\delta = \mathcal{F}1_\delta)$$

$$U, V sets. \quad H(\mathcal{S}1_{\mu G}|\mathcal{F}1_\mu[U/V]) = \forall \mathcal{S}_\delta|\mathcal{F}_\delta : \{\{D^\delta\}\} \in H, \mathcal{S}1_\delta \in \mathcal{S}1_{\mu G}, \mathcal{F}1_\delta \in \mathcal{F}1_{\mu G}. (\mathcal{S}_\delta = \mathcal{S}1_\delta \wedge \mathcal{F}_\delta = \mathcal{F}1_\delta[U/V])$$

The rules of $\lambda\mu_r$PRL can be found in figure 6.2.

Proposition 6.5. *A $\lambda\mu$PRL-term corresponds to a proof in $\lambda\mu_r$PRL iff it is μ-safe.*

Proof. $\Rightarrow$) We have to show that for any $\lambda\mu_r$PRL-term t for any subterm of the form $\lambda x.u$ holds: for any μ-variable δ occurring freely in u, $x \notin S_\delta$.

We know that in $\lambda\mu_r$PRL the application of the rule `lambdaI` would have added x to the set of forbidden variables for all μ-variables. If any of the passive formulae associated with these μ-variables was activated in the subsequent proof, this would result in a subterm in which the respective μ-variable would occur freely. The restriction of the rule `hypothesis` of $\lambda\mu_r$PRL would prevent the use of x as long as no other context formula is activated and thus it cannot occur freely in the subterm.

$\Leftarrow$) If a $\lambda\mu$PRL-term t is μ-safe, we know that for any subterm of the form $\lambda x.u$ for any μ-variable δ occurring freely in u , $x \notin S_\delta$. Therefore in none of the subproofs corresponding to subterms of the form $[\delta]v$ of $\lambda x.u$, x is used by the `hypothesis`-rule, since in that case it would occur freely in v. Accordingly the restriction put on $\lambda\mu$PRL terms by the rules `lambdaI` and `hypothesis` are met and the term is also a term of $\lambda\mu_r$PRL. □

It remains to show that $\lambda\mu_r$PRL is in fact correct and complete for full intuitionistic propositional logic. E.g. this could be done by simulating its rules in Nuprl or a calculus equivalent to its propositional subsystem.

Proposition 6.6. *A proposition ϕ is provable in $\lambda\mu_r$PRL iff it is provable in the constructive logic of the Nuprl proof development system.*

Proof. $\Rightarrow$) We simulate the sequents of$\lambda\mu_r$PRL such that the set of forbidden variables $\mathcal{F}$ "splits" the sequent into several sequents which themselves can be seen as implications: One sequent for each μ-variable and one for the active conclusion. These sequents are connected by a constructive $\vee$ and the antecedent of each "subsequent" contains merely the hypotheses which are not contained in $\mathcal{F}$ of the respective succedent. E.g. the sequent

$$\{y\} : \{\{D^\alpha\}\}, x : A, y : B \vdash \{x\} : C$$

of $\lambda\mu_r$PRL induces the split sequents $(A \vdash D)$ and $(B \vdash C)$ and is simulated by

$$\vdash (A \supset D) \vee (B \supset C)$$

in Nuprl. Then we can proof all rules of $\lambda\mu_r$PRL as lemmas in Nuprl.

The interesting cases are the rules `lambdaI`, `mu1`, `abort1` and `hypothesis`, although the simulation renders them to very simple constructive theorems. The rest of the rules

---

G, $x : T$, H $\vdash \{x\}|\mathcal{F} : T$ ext x
BY hypothesis $\$i$
$if\ x \notin \mathcal{F}$

(no subgoals)

H($\mathcal{S}_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S} \setminus \{x\}|\mathcal{F} : A \supset B$ ext $\lambda x.b$
BY lambdaI x

H ($\mathcal{S}_{\mu H}|\mathcal{F}_{\mu H} \cup \{x\}$), x:A $\vdash \mathcal{S}|\mathcal{F} : B$ ext b

G($\mathcal{S}1_{\mu G} \cup \mathcal{S}2_{\mu G}|\mathcal{F}_{\mu H}$), pf:$A \supset B$, H($\mathcal{S}1_{\mu H} \cup \mathcal{S}2_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}_1 \cup \mathcal{S}_2|\mathcal{F}$: C ext $b[pf\ a/y]$
BY functionE $\$i$

G($\mathcal{S}1_{\mu G}|\mathcal{F}_{\mu G}$), pf:$A \supset B$, H($\mathcal{S}1_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}_1|\mathcal{F}$: A ext a
G($\mathcal{S}2_{\mu G}|\mathcal{F}_{\mu G}$), y:B, H($\mathcal{S}2_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}_2|\mathcal{F}$: C ext b

---

H $\vdash \mathcal{S}_\alpha|\mathcal{F} : B$ ext $\mu\_\alpha.b$
BY mu1 α

H , $\mathcal{S}_\alpha|\mathcal{F} : \{\{B^\alpha\}\} \vdash \mathcal{S}|\mathcal{F} : \bot$ ext b

G ,$\mathcal{S}|\mathcal{F}_\alpha : \{\{B^\alpha\}\}$, H$\vdash$ $\emptyset|\mathcal{F} : \bot$ ext $[\alpha]$ b
BY abort1 $\$i$

G , H $\vdash \mathcal{S}|\mathcal{F}_\alpha : B$ ext b

H $\vdash \mathcal{S}_\alpha|\mathcal{F} : B$ ext $\mu\_\alpha.b$
BY mu2 α

H $\vdash \mathcal{S}|\mathcal{F} : \bot$ ext b

G ,$\mathcal{S}_\alpha \cup \mathcal{S}|\mathcal{F}_\alpha : \{\{B^\alpha\}\}$, H $\vdash \emptyset|\mathcal{F} : \bot$ ext $[\alpha]$ b
BY abort2 $\$i$

G , $\mathcal{S}_\alpha|\mathcal{F}_\alpha : \{\{B^\alpha\}\}$, H $\vdash \mathcal{S}|\mathcal{F}_\alpha : B$ ext b

---

H($\mathcal{S}1_{\mu H} \cup \mathcal{S}2_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}_1 \cup \mathcal{S}_2|\mathcal{F} : A \wedge B$ ext $\langle a, b\rangle$
BY andI

H($\mathcal{S}1_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}_1|\mathcal{F} : A$ ext a
H($\mathcal{S}2_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}_2|\mathcal{F} : B$ ext b

G, z:$A \wedge B$, H($\mathcal{S}_{\mu H}|\mathcal{F}_{\mu H}$) $\vdash \mathcal{S}[\{z\}/\{a,b\}]|\mathcal{F}$:C ext $let\ \langle a, b\rangle = z\ in\ u$
BY andE $\$i$

G , $a : A$, $b : B$, H($\mathcal{S}_{\mu H}|\mathcal{F}_{\mu H}[\{a,b\}/\{z\}]$) $\vdash \mathcal{S}|\mathcal{F}[\{a,b\}/\{z\}]$:C ext u

---

H $\vdash \mathcal{S}|\mathcal{F} : A \vee B$ ext $inl(a)$
BY orI1

H $\vdash \mathcal{S}|\mathcal{F} : A$ ext a

H $\vdash \mathcal{S}|\mathcal{F} : A \wedge B$ ext $inr(b)$
BY orI2

H $\vdash \mathcal{S}|\mathcal{F} : B$ ext b

G($\mathcal{S}1_{\mu G}[\{z\}/\{a\}] \cup \mathcal{S}2_{\mu G}[\{z\}/\{b\}]|\mathcal{F}_{\mu G}$), z:$A \vee B$, H($\mathcal{S}1_{\mu H}[\{z\}/\{a\}] \cup \mathcal{S}2_{\mu H}[\{z\}/\{b\}]|\mathcal{F}_{\mu G}$) $\vdash \mathcal{S}_1[\{z\}/\{a\}] \cup \mathcal{S}_2[\{z\}/\{b\}]|\mathcal{F}$:C ext $case\ z\ of\ inl(a) : u\ |\ inr(b) : v$
BY orE $\$i$

G($\mathcal{S}1_{\mu G}|\mathcal{F}_{\mu G}$) , $a : A$, H($\mathcal{S}1_{\mu H}|\mathcal{F}_{\mu H}[\{a\}/\{z\}]$) $\vdash \mathcal{S}_1|\mathcal{F}[\{a\}/\{z\}]$:C ext u
G($\mathcal{S}2_{\mu G}|\mathcal{F}_{\mu G}$) , $b : B$, H($\mathcal{S}2_{\mu H}|\mathcal{F}_{\mu H}[\{b\}/\{z\}]$) $\vdash \mathcal{S}_2|\mathcal{F}[\{b\}/\{z\}]$:C ext v

---

Figure 6.1: The inference rules of $\lambda\mu_r$PRL

do not alter the $\mathcal{F}$-sets of the sequent (apart from replacing the variable associated with an eliminated conjunction or disjunction by the new variable-names of the conjuncts or disjuncts) and are obviously constructive.

We omit detailed proofs in Nuprl or another constructive logic since they are trivial and in no way illuminating.

To describe a set of hypotheses without the hypotheses forbidden by $\mathcal{F}$ we write $H \setminus \mathcal{F}$. The formulation of the rule lemmas may ignore the static part of the rule, i.e. the part of the μ-contexts which are not altered by an application of the rule.

- `hypothesis`: Is simulated by simply $\vdash A \supset A$ since the condition $x \notin \mathcal{F}$ induces a sequent which does contain A as hypothesis. We could also write the lemma as disjunction between all sequents induced by the μ-variables and their accompanying sets of forbidden λ-variables. But as they are not used in the rule we can omit them (If there were a disjunction, the proof would require to choose the disjunct $A \supset A$ anyway.) Thus the rule can be proven by Nuprl's rule `hypothesis`.

- `lambdaI`: Simulated by $\vdash (((H \wedge A) \supset B) \vee (H \supset \Delta)) \supset ((H \supset (A \supset B)) \vee (H \supset \Delta))$ where Δ represents the μ-formulae of the sequent and $H \supset \Delta$ represents a disjunction of implications, such that for every μ-variable δ of Type D with the set of forbidden λ-variables F_δ the respective implication is $H \setminus \mathcal{F}_\delta \supset D$. A does not occur in these implications, since it is added to the forbidden set for each μ-variable. As a consequence the part $H \supset \Delta$ can be omitted in the rule-lemma. However, $\vdash ((H \wedge A) \supset B) \supset (H \supset (A \supset B))$ is exactly the usual introduction rule for an implication.

- `mu1`: $H' \equiv H \setminus \mathcal{F}_\alpha$. Then the rule is simulated by $\vdash ((H' \supset \bot) \vee (H' \supset B)) \supset (H' \supset B)$ which is obviously constructively valid.

- `abort1`: $H' \equiv (G \cup H) \setminus \mathcal{F}_\alpha$ and $H'' \equiv (G \cup H) \setminus \mathcal{F}$. Then the rule is simulated by $\vdash (H' \supset B) \supset (H' \supset B) \vee (H'' \supset \bot)$ which is also obviously constructively valid.

$\Leftarrow$) The propositional inference rules of Nuprl's logic (or an equivalent constructive logic) are very obviously provable in $\lambda\mu_r$PRL as there is no need for context formulae and thus the set F trivially remains empty throughout the proofs of the rules.

□

Now we can use the concept of μ-safety for our original purpose. Evaluating a term t of $\lambda\mu$PRL to decide its constructivity consists of two steps:

1. computing the scopes of the free μ-variables present in t and
2. checking for every subterm of the form $\lambda x.u$ that x is not in the scope of any free μ-variable occurring in u.

Remark We could also verify that the set of μ-safe $\lambda\mu$PRL-terms is closed under reduction.

The last question remaining is whether we can safely use Murthy's evaluation to compute the constructive λ-term from a constructive $\lambda\mu$PRL-term.

6.3 Constructive Content of λμPRL-terms

It might be desirable to recover the equivalent λ-term of a constructive λμ-term. Under this aspect, it seems sensible to consider whether we can use Murthy's evaluation [Mur90] for this purpose. As described above, we still have to ensure, that the elimination rules for ∨ and ∧ are applied directly after the ⊃-introduction of the respective hypothesis. But under this prerequisite, we can again use the notion of μ-safe terms: We can show for the set of μ-safe λμPRL-terms, that reducing them by a combination of ϵ-reduction and a deterministic evaluation strategy, the resulting term does not contain any free, undefined variables, i.e. the resulting term is constructive.

Being interested in the "pure" (as opposed to λμ) λ-term, the straightforward approach is to start with the outermost occurrence of μ and to evaluate each μ-abstraction contained in the term directly followed by evaluating the occurrences of the abort-operator $\mathcal{A}$ produced by the evaluation of μ. Like Murthy, we use a weak-head-reduction and due to the non-determinism induced by the two principal subterms of the pair-construct, we have to fix an evaluation strategy. This means either left-to-right or right-to-left evaluation order. However, with μ-safe terms it does not matter which subterm is chosen first (the other subterm possibly being thrown away by $\mathcal{A}$), since both variants must result in a constructive term. Still, this hints a pathological case which we will briefly address in the next section.

Therefore let the new evaluation work as follows:

Definition 6.7. *(c-evaluation) Given a λμPRL-term t:*

1. *Choose the outermost μ-abstraction contained in t and evaluate it by the rule* $C[\mu\alpha.u] \rightsquigarrow_\epsilon u[\mathcal{A}(C[v])/[\alpha]v]$.
2. *Evaluate every occurrence of* $\mathcal{A}$ *in the resulting term left-to-right by* $C[\mathcal{A}(u)] \rightsquigarrow_\epsilon u$.
3. *Repeat these two steps as long as there are occurrences of μ in the resulting term.*

We denote this adapted ϵ-reduction as c-evaluation and write $t \rightsquigarrow_c t'$ *if* t' *is obtained by c-evaluating t..*

Proposition 6.8. *Applying c-evaluation to a μ-safe λμPRL-term results in a closed term, i.e. for* t, t' *terms of λμPRL, if t is μ-safe and* $t \rightsquigarrow_c t'$ *then* t' *contains no free variables.*

Proof. We know that t is μ-safe. Following from the definition, every subterm of t is μ-safe as well. When we apply c-reduction to t, it is split into a context $C[\bullet]$ with a hole and a term of the form $\mu\alpha.u$. We have two cases:

1. α does not occur freely in u: Then $C[\mu\alpha.u] \rightsquigarrow_c u$. Then u is μ-safe because t is μ-safe and u is a subterm of t.
2. α occurs freely in u: $C[\mu\alpha.u]$ is first reduced to $u[\mathcal{A}(C[s])/[\alpha]s]$, then – by evaluating each occurrence of $\mathcal{A}$ – to the leftmost, innermost $C[v]$. Since t is μ-safe, we know that for any λ-variable x there are three cases:
   (a) v has a subterm $\lambda x.w$ by which any x occurring in v is bound. Then x is also bound in t', since v is not changed in the evaluation (for it is the innermost occurrence of a subterm $[\alpha]s$).

(b) v does not contain x then it does not occur in in v after the evaluation either.

(c) x is bound in $C[\bullet]$ where α is not free. Since c-evaluation replaces every subterm of the form $[\alpha]w$ by $\mathcal{A}(C[w])$, t' is one of these terms $C[w]$ and therefore still contains a subterm $\lambda x.s$ where s is a superterm of w, so that every occurrence of x in w is bound. Occurrences of x in $C[\bullet]$ also (obviously) remain bound, since the context is not altered but merely moved during c-evaluation.

□

6.4 Non-μ-safe Terms with Constructive Content

In section 6.1 we briefly mentioned that a deterministic evaluation in presence of the non-determinism intrinsic to classical logic might "loose" constructive content. To conclude this chapter, we would like to elaborate this point, as the terms which are concerned are not μ-safe and as such have not been considered up to now.

The first question is, in which case a term may evaluate to a constructive term or a non-constructive term depending on the evaluation order in which ϵ-reduction is applied. For $\lambda\mu$PRL-terms, this situation might occur because the pair-construct contains two principal subterms.

If we have some term context with a μ-abstraction $C[\mu\alpha.t]$ and a pair $\langle u, v\rangle$ as subterm of t where u contains a subterm $[\alpha]u'$ and v a subterm $[\alpha]v'$, it depends on the evaluation order whether the resulting term is $C[u']$ or $C[v']$ (provided there are no other μ-abstractions which are evaluated first and there are no other free occurrences of α). Now $C[u']$ might be constructive whereas $C[v']$ is not.

Let us take a look at an admittedly highly artificial example.

Example 6.3. Consider the proposition $\phi \equiv A \supset ((A \vee B) \wedge ((A \vee \neg A) \vee B))$. A witness of this proposition is the term

$$t \equiv \lambda x.\mu\alpha.[\alpha]\left\langle inr(\mu\beta.[\alpha]\left\langle inl(x), inl(inl(x))\right\rangle, inr(\mu\delta.[\alpha]\left\langle inl(x), inl(lem)\right\rangle\right\rangle$$

where *lem* is a proof term of the law of the excluded middle. We omit the complete proof since it is rather lengthy and uninteresting. As we know, the term *lem* is not μ-safe, and consequently t is not μ-safe either. However, if we evaluate t by c-evaluation, we obtain the term $\lambda x.\langle inl(x), inl(inl(x))\rangle$ which corresponds to an intuitionistic proof of ϕ. On the contrary, if the subterms of the crucial pair in t had been reverted, then the result would have been a non-closed term.

Obviously, if we look at the term tree of such terms and transfer the concept of μ-safety to such trees, they contain partial trees which are μ-safe. Accordingly, we could define a notion of *partial μ-safety*.

Definition 6.9. *(partial μ-safety) Partial μ-safety is defined recursively as follows:*

- *A $\lambda\mu$PRL term t which does not contain any subterm of the form $\langle p_1, p_2\rangle$ is partially μ-safe, if it is μ-safe.*
- *A $\lambda\mu$PRL term $t \equiv C[\langle p_1, p_2\rangle]$ where $C[\bullet]$ does not contain any pair-constructors, is partially μ-safe, if it is μ-safe or either $C[p_1]$ or $C[p_2]$ is partially μ-safe.*

We conjecture, that every partially μ-safe term can be evaluated to a constructive term.

In this chapter, we have shown, that we cannot only distinguish between constructive and non-constructive $\lambda\mu$PRL-terms by identifying the use of `abort`-rules which would exclude many terms which in fact do have constructive content. Based on a more detailed analysis of the term structure, the resulting notions of μ-safe and partially μ-safe terms can easily be used to implement a procedure which identifies terms with constructive content.

Remark Still, we certainly cannot conclude from a term which evaluates as non-constructive, that there is no constructive proof of the proposition! The result simply states, that the proof in question relies on a classical argument.

Chapter 7

$\lambda\mu$PRL and $PROG_K$

This chapter is concerned with the relation between $\lambda\mu$PRL and Murthy's system $PROG_K$ [Mur91b].

Like $\lambda\mu$PRL, $PROG_K$ has been developed in the context of the Nuprl Proof development system and provides an extension to classical logic. Moreover, Murthy's results together with the work of Griffin [Gri90] were pioneering in the research of computational content of classical proofs. Parigot's original $\lambda\mu$-calculus [Par92] was also inspired by Griffin and Murthy. Therefore it is interesting to look into the exact relation between $\lambda\mu$PRL and $PROG_K$.

If we both restrict and extend $\lambda\mu$PRL in a convenient way, it is possible to reproduce Murthy's results as to the extraction of computational content.

For a short introduction to $PROG_K$, see chapter 2.3.

7.1 $\lambda\mu$PRL and $PROG_K$

As mentioned above, $\lambda\mu$PRL and $PROG_K$ are set in a similar context. Consequently, there is some common ground between the two:

- both use logical rules of Nuprl
- both are top-down refinement calculi
- both are extended to classical logic by an additional rule
- application of the classical rule is signalled in the proof term by a special operator
- both are minimal logics, i.e. there is no explicit elimination rule for $\bot$.

Still, there are differences as well.

The intention behind $PROG_K$ is the extraction of constructive content from classical proofs. It is a calculus for Peano Arithmetic and therefore contains rules for universal and existential quantification and arithmetical expressions. At the same time $\mathbb{N}$ is the only data type. Because of problems with the axiom of choice (which will be briefly addressed in chapter 8) theorems are restricted to the decidable fragment Σ_1^0 (respectively Π_2^0). In this fragment classical logic is conservative over intuitionistic logic.

$\lambda\mu$PRL is a calculus for propositional logic only, but without restrictions concerning data types or theorems. This reflects the idea behind $\lambda\mu$PRL to let the user decide whether he wants to employ classical logic or not (i.e. needs constructive content or not). Accordingly, proofs in $\lambda\mu$PRL do not necessarily contain constructive content. Still, to some extend it is possible to distinguish between constructive and non-constructive proofs (as described in chapter 6).

Murthy extends the intuitionistic calculus $PROG_J$ to its classical version $PROG_K$ by the double negation elimination rule. The associated operator is Felleisen's $\mathcal{C}$ [FFKD87] and "classical" hypotheses are not distinguished from "constructive" ones as λ is the only binding operator. When reducing $\mathcal{C}$, another new operator is used: The abort-operator $\mathcal{A}$ (also due to Felleisen).

$\lambda\mu$PRL is classical because of the `abort`-rules and "classical" hypotheses are syntactically different from "constructive" ones. Their associated variables are bound by the additional μ-operator and their usage is signalled by the $[-]$-operator.

For $\lambda\mu$PRL we settled on a confluent call-by-name $\beta\mu$-reduction. Reduction in $PROG_K$ is defined in terms of evaluation contexts. The one-step-evaluation rules for the operators $\mathcal{C}$ and $\mathcal{A}$ are adapted from Felleisen's $\lambda_{\mathcal{C}}$-calculus. Thus $\mathcal{C}$ works like a control operator in a not purely functional programming language. This kind of evaluation corresponds to the symmetric reduction rule of the $\lambda\mu\mu'$-calculus described in chapter 5. Accordingly the inherent non-determinism causes the well-known problem with confluence. To solve this problem, Murthy defines a deterministic evaluation strategy. So, if a term contains two different values, the evaluation strategy determines which one is "chosen".

Since we did not use the symmetric reduction rule for $\lambda\mu$PRL, it is quite obvious that reduction will not be preserved in a translation between the two systems. However, in section 7.3 we will consider how we could adapt $\lambda\mu$PRL for this purpose.

Apart from the differences, we can show a strong relation between $\lambda\mu$PRL and $PROG_K$. This can be done in a very similar way to how de Groote [dG94b] relates a first order version of Parigot's $\lambda\mu$ to a subtheory of Felleisen's syntactic theory of sequential control (for short: λ_C) [FFKD87, FH92]. So the next section will be concerned with the respective translations between $\lambda\mu$PRL and $PROG_K$.

7.2 Translation between $\lambda\mu$PRL and $PROG_K$

As mentioned before, $\lambda\mu$PRL and $PROG_K$ do not cover the same range of formulae. Since an extension of $\lambda\mu$PRL to arithmetic and first order logic is desirable, we will define a variant $\lambda\mu_c$PRL in this section. This will extend the logical language of $\lambda\mu$PRL to include the arithmetical and first-order expressions of $PROG_K$ as well as the accompanying rules. To achieve further comparability with $PROG_K$, we will restrict $\lambda\mu_c$PRL to decidable formulae (i.e. Σ^0_1 and Π^0_2 sentences) and to the type $\mathbb{N}$. As a result $\lambda\mu_c$PRL is a calculus for Peano Arithmetic, too. The terms and additional rules of $\lambda\mu_c$PRL can be found in figure 7.1.

Remark Note that we also add the abort-operator $\mathcal{A}$ to the term language of $\lambda\mu_c$PRL. This necessity will become clearer in the context of reduction.

For soundness reasons, it is also required to include a call-by-value λ-abstraction for the arithmetic rules having "axiom" as their extract term. This guarantees

that the subterm does not cause any control side-effects (otherwise the context might be discarded during evaluation, so that the result is not "axiom").

Another property of $PROG_K$ is that sometimes integer expressions are distinguished from propositions. This is achieved by annotating them as $-^D$.

Since our focus in this chapter is on the relation of $\lambda\mu_c$PRL and $PROG_K$, we will not go into further detail on the non-propositional part of the two calculi. After all the additional rules and terms of $\lambda\mu_c$PRL (in comparison to $\lambda\mu$PRL) are identical to those of $PROG_K$. The crucial differences between the two calculi arise from the "classical" constructs μ and $\mathcal{C}$. Therefore it is sufficient to merely consider the propositional fragments for our current purpose.

On this basis we can define translations from $\lambda\mu_c$PRL-terms into the terms of $PROG_K$ and reversely. This can be done quite straightforwardly, since the only difference between $\lambda\mu_c$PRL and $PROG_K$ are the terms μ and $[-]$ on the one hand and $\mathcal{C}$ on the other and their respective typing rules. Since we adapted the non-propositional rules of $PROG_K$ to $\lambda\mu_c$PRL, these are identical anyway. Consequently, we will be content to take a look at the propositional fragment. The proceeding is basically an extension of de Groote's in [dG94b].

We start with defining a $\mathcal{C}$-transform of a $\lambda\mu_c$PRL-term (for the reasons stated above, we omit the non-propositional fragment).

Definition 7.1. *The $\mathcal{C}$-transform $[\![t]\!]^{\mathcal{C}}$ of a $\lambda\mu_c$PRL-term t is defined inductively by:*

1. $[\![x]\!]^{\mathcal{C}} = x$;
2. $[\![\lambda x.t]\!]^{\mathcal{C}} = \lambda x.[\![t]\!]^{\mathcal{C}}$;
3. $[\![(s\ t)]\!]^{\mathcal{C}} = ([\![s]\!]^{\mathcal{C}}\ [\![t]\!]^{\mathcal{C}})$;
4. $[\![\langle s,t\rangle]\!]^{\mathcal{C}} = \langle [\![s]\!]^{\mathcal{C}}, [\![t]\!]^{\mathcal{C}}\rangle$;
5. $[\![\texttt{let}\ \langle x,y\rangle = z\ \texttt{in}\ u]\!]^{\mathcal{C}} = spread([\![z]\!]^{\mathcal{C}}; x, z.[\![u]\!]^{\mathcal{C}})$;
6. $[\![inl(t)]\!]^{\mathcal{C}} = inl([\![t]\!]^{\mathcal{C}})$;
7. $[\![inr(t)]\!]^{\mathcal{C}} = inr([\![t]\!]^{\mathcal{C}})$;
8. $[\![\texttt{case}\ z\ \texttt{of}\ \texttt{inl}(x) : u \mid \texttt{inr}(y) : v]\!]^{\mathcal{C}} = decide([\![z]\!]^{\mathcal{C}}; x.[\![u]\!]^{\mathcal{C}}; y.[\![v]\!]^{\mathcal{C}})$;
9. $[\![\mu\alpha.t]\!]^{\mathcal{C}} = (\mathcal{C}\ \lambda\alpha.[\![t]\!]^{\mathcal{C}})$;
10. $[\![[\alpha]t]\!]^{\mathcal{C}} = (\alpha\ [\![t]\!]^{\mathcal{C}})$;
11. $[\![(\mathcal{A}\ t)]\!]^{\mathcal{C}} = (\mathcal{A}\ [\![t]\!]^{\mathcal{C}})$;

Based on this definition we can show the following:

Proposition 7.2. *If t is a $\lambda\mu_c$PRL-term and A is a simple type such that $\vdash_\mu A$ $\texttt{ext}$ t then $\vdash_{\mathcal{C}} A$ $\texttt{ext}$ $[\![t]\!]^{\mathcal{C}}$.*

$\lambda\mu_c$PRL-terms:

$$t ::== x \mid \lambda x.t \mid (t\ t) \mid \mu\alpha.t \mid [\alpha]\,t \mid \langle t,t\rangle \mid let\ \langle x,y\rangle = z\ in\ t \mid$$
$$inl(t) \mid inr(t) \mid case\ t\ of\ in_1(x) : t\ or\ in_2(x) : t$$
$$\mathcal{A}\ t \mid \lambda^v x.t \mid (t\ t)_v \mid axiom \mid 0 \mid ind(t;t;x,y.t) \mid succ(t) \mid t+t \mid t\times t$$

The rules for $\supset$,$\wedge$ and $\vee$ the rule `hypothesis` are the same as in $\lambda\mu$PRL.

Arithmetical axioms

$n : \mathbb{N}, m : \mathbb{N}, u : n = m \vdash m = n$ `ext` $(\lambda^v u'.axiom\ u)_v$
`BY symmetry` $\$i$

$a : \mathbb{N}, b : \mathbb{N}, c : \mathbb{N}, u : a = b, v : b = c \vdash a = c$ `ext` $((\lambda^v u', v'.axiom\ u)_v\ v)_v$
`BY transitivity` $\$i$

$n : \mathbb{N}, u : succ(n) = 0 \vdash \bot$ `ext` $(\lambda^v u'.axiom\ u)_v$
`BY succMonotonicity` $\$i$

$x : \mathbb{N}, y : \mathbb{N}, u : x^D = y^D \vdash succ(x^D) = succ(y^D)$ `ext` $(\lambda^v u'.axiom\ u)_v$
`BY succInjectivity` $\$i$

$n : \mathbb{N}, m : \mathbb{N}, u : succ(x^D) = succ(y^D) \vdash x^D = y^D$ `ext` $(\lambda^v u'.axiom\ u)_v$
`BY succSurjectivity` $\$i$

Rules for arithmetic and quantification

$n^D : \mathbb{N} \vdash T$ `ext` $ind(n^D; B; i, n^D.F(n-1)^D(i))$
`BY induction` $\$i$

$n^D : \mathbb{N} \vdash T[0^D/n^D]$ `ext` B
$n^D : \mathbb{N} \vdash \forall m^D : \mathbb{N}.T[m^D/n^D] \supset T[succ(m)^D/n^D]$ `ext` F

$H \vdash \forall n^D : \mathbb{N}.T$ `ext` $\lambda x^D.b$
`BY allI` x

$H, x^D : \mathbb{N} \vdash T[x^D/n^D]$ `ext` b

$G, pf : \forall n^D : \mathbb{N}.T, H \vdash C$ `ext` $b[pf\ t^D/y]$
`BY allE` $\$i\ y$

$G, x^D : \mathbb{N}, H \vdash t^D \in \mathbb{N}$ `ext` $axiom$
$G, y : T[t^D/n^D], H \vdash C$ `ext` b

$H \vdash \exists n^D : \mathbb{N}.T$ `ext` $< t^D, b >$
`BY exI` t^D

$H \vdash t^D \in \mathbb{N}$ `ext` $axiom$
$H \vdash T[t^D/n^D]$ `ext` b

$G, z : \exists n^D : \mathbb{N}.T, H \vdash C$ `ext let` $< x^D, b > = z$ `in` u
`BY exE` $\$i\ x\ b$

$G, x^d : \mathbb{N}, b : T[x^D/n^D], H[x^D/n^D] \vdash C[x^D/n^D]$ `ext` u

---

Figure 7.1: The calculus $\lambda\mu_c$PRL for Peano Arithmetic

Proof. If we interpret every sequent

$$\Gamma \vdash A \texttt{ ext } t$$

of $\lambda\mu_c$PRL as

$$\Gamma^\neg \vdash A \texttt{ ext } t$$

where $\Gamma^\neg$ arises from Γ by replacing every μ-declaration of the form $\{\{B^\beta\}\}$ by a λ-declaration $\beta : \neg B$ (this is possible since the λ- and μ-variables come from the same alphabet), we are able to simulate the rules `mu1`, `mu2`, `abort1` and `abort2` in $PROG_K$:

$\Gamma^\neg \vdash A$ `ext` $(\mathcal{C}\ \lambda\alpha.t)$
by double negation elimination
|_ $\Gamma^\neg \vdash \neg\neg A$ `ext` $\lambda\alpha.t$
by function-intro
|_ $\Gamma^\neg, \alpha : \neg A \vdash \bot$ `ext` t

$\Gamma^\neg, \alpha : \neg A \vdash \bot$ `ext` $(\alpha\ t)$[1]
by function-elim
|_ $\Gamma^\neg \vdash A$ `ext` t
|_ $\Gamma^\neg, \alpha : \neg A, z : \bot \vdash \bot$ `ext` z
by hypothesis

The extract terms of the rule-simulations match the translation given above.

Apart from the `mu-` and `abort-` rules, the rules of $PROG_K$ and $\lambda\mu_c$PRL are almost identical for their respective logical connectives (but for syntactical differences). Consequently the simulation is straightforward. □

The inverse translation is based on the following proof of the double negation elimination rule in $\lambda\mu_c$PRL:

$\vdash \neg\neg A \supset A$ `ext` $\lambda x.\mu\alpha.(x\ \lambda y.[\alpha]y)$
by lambdaI
|_ $\neg\neg A \vdash A$ `ext` $\mu\alpha.(x\ \lambda y.[\alpha]y)$
by mu1 α
|_ $x : \neg\neg A, \{\{A^\alpha\}\} \vdash \bot$ `ext` $(x\ \lambda y.[\alpha]y)$
by functionE 1
| |_ $\{\{A^\alpha\}\} \vdash \neg A$ `ext` $\lambda y.[\alpha]y$
| by lambdaI
| |_ $y : A, \{\{A^\alpha\}\} \vdash \bot$ `ext` $[\alpha]y$
| by abort2 2
| |_ $y : A, \{\{A^\alpha\}\} \vdash A$ `ext` y
| by hypothesis
|_ $z : \bot, \{\{A^\alpha\}\} \vdash \bot$ `ext` z
by hypothesis

Applying the extract term of the proof above to the translated subterm t now yields the μ-transform of a $PROG_K$-term $(\mathcal{C}\ t)$. Now we can proceed to give the translation from $PROG_K$ to $\lambda\mu_c$PRL.

[1] which is the result of applying the substitution in $z[(\alpha\ t)/z]$

Definition 7.3. *The μ-transform $[\![t]\!]^\mu$ of a $PROG_K$- term t is defined inductively by:*

1. $[\![x]\!]^\mu = x$;
2. $[\![\lambda x.t]\!]^\mu = \lambda x.[\![t]\!]^\mu$;
3. $[\![(s\ t)]\!]^\mu = ([\![s]\!]^\mu\ [\![t]\!]^\mu)$;
4. $[\![\langle s,t\rangle]\!]^\mu = \langle [\![s]\!]^\mu, [\![t]\!]^\mu\rangle$;
5. $[\![\texttt{let}\ \langle x,y\rangle = z\ \texttt{in}\ u]\!]^\mu = spread([\![z]\!]^\mu; x, z.[\![u]\!]^\mu)$;
6. $[\![inl(t)]\!]^\mu = inl([\![t]\!]^\mu)$;
7. $[\![inr(t)]\!]^\mu = inr([\![t]\!]^\mu)$;
8. $[\![\texttt{case}\ z\ \texttt{of}\ \texttt{inl}(x) : u \mid \texttt{inr}(y) : v]\!]^\mu = decide([\![z]\!]^\mu; x.[\![u]\!]^\mu; y.[\![v]\!]^\mu)$;
9. $[\![(\mathcal{C}\ t)]\!]^\mu = \mu\alpha.([\![t]\!]^\mu\ \lambda y.[\alpha]y)$;
10. $[\![(\mathcal{A}\ t)]\!]^\mu = (\mathcal{A}\ [\![t]\!]^\mu)$;

Based on this definition we can show the following:

Proposition 7.4. *If t is a $PROG_K$-term and A is a simple type such that $\vdash_{\mathcal{C}} A\ \texttt{ext}\ t$ then $\vdash_\mu A\ \texttt{ext}\ [\![t]\!]^\mu$.*

Proof. This can be shown by an induction on the derivation of $\vdash_{\mathcal{C}} A\ \texttt{ext}\ t$. □

Remark A translation of the operator $\mathcal{A}$ is not necessary for the translation of derivations in the two calculi (since it may not occur in a derivation). However, it becomes interesting when we take a look at reduction.

In this section we have established translations between and $PROG_K$ and $\lambda\mu_c$PRL which preserve typing. The next section will concern the question whether and how the reduction relations of the two calculi correspond.

7.3 Preservation of Reduction

When relating two calculi and giving a translation, usually the question of reduction preservation arises. If we compare the reduction rules of $PROG_K$ to the $\beta\mu$-reduction defined in chapter 5, it is rather obvious, that reduction cannot be preserved in any direction. In fact even the notion is different. Reduction in $PROG_K$ is an evaluation depending on evaluation contexts and a deterministic strategy - as opposed to a reduction where the rules can applied to any redex.

However, the usual intuitionistic β-reduction-rules are the same in both $\lambda\mu$PRL and $PROG_K$. The difference is caused by the rather restricted reduction of the μ-operator in $\lambda\mu_c$PRL. As is known, this restriction is necessary for the confluence of the reduction relation. In $PROG_K$ such a restriction is futile, since Murthy defines a deterministic reduction strategy. While a term may contain more than one value, the evaluation strategy will always return the same one.

So there is also a practical reason to compare the respective reduction relations: After all the ability to extract values from $\lambda\mu_c$PRL-terms might be useful. So we would like to consider another notion of reduction for $\lambda\mu_c$PRL. Instead of the μ-reduction-rules defined in chapter 5, we will define two μ_c-rules.

The new rules are based on those for $\mathcal{C}$ and $\mathcal{A}$ in $PROG_K$. We denote Murthy's evaluation as c-reduction. It is defined in the using a term context $E[-]$ which (as usual) denotes an arbitrary term of $PROG_K$ with a hole:

Definition 7.5. *(c-reduction)*

$$\begin{array}{lll} E[(\mathcal{C}\ t)] & \rightsquigarrow_c & (t\ \lambda x.(\mathcal{A}\ E[x])) \\ E[(\mathcal{A}\ t)] & \rightsquigarrow_c & t \end{array}$$

Now we define μ_c-reduction in a similar way. The adaption is induced by the translation of the respective terms from $PROG_K$ to $\lambda\mu_c$PRL. This is why we originally added the $\mathcal{A}$-operator to the term-language of $\lambda\mu_c$PRL.

Definition 7.6. *(μ_c-reduction)*

$$\begin{array}{lll} E[\mu\alpha.t] & \rightsquigarrow_{\mu_c} & t[\mathcal{A}(E[u])/[\alpha]u] \\ E[\mathcal{A}(t)] & \rightsquigarrow_{\mu_c} & t \end{array}$$

Because of the known confluence-issues, the above "reduction" requires an evaluation strategy (the reduction rules may not be applied independently of their context). Thus we also adopt Murthy's deterministic evaluation strategy which is defined in chapter 2.3. This makes sure the rules are applied in the same order when we compare the evaluation in the two calculi.

Now we can establish, that the translations defined in the previous section do preserve the results of evaluation (although not necessarily in the same number of reduction steps).

We denote the evaluation of $\lambda\mu_c$PRL that combines Murthy's reduction strategy with β- and μ_c-reduction by $ev_{\overline{\mu}\rightarrow}$ and the evaluation of $PROG_K$ by $ev_{\overline{c}\rightarrow}$.

Proposition 7.7. *Let t_1 and t_2 be $\lambda\mu_c$PRL-terms. If $t_1\ ev_{\overline{\mu}\rightarrow}\ t_2$ then $[\![t_1]\!]^{\mathcal{C}}\ ev_{\overline{c}\rightarrow}\ [\![t_2]\!]^{\mathcal{C}}$.*

Proof. This result is obvious for the β-rules since the differences in these cases are purely syntactical.

The interesting cases concern the structural rules.

Instead of presenting a case split on every possible evaluation context, we sketch the idea by writing $E^{\mathcal{C}}[-]$ for the $\mathcal{C}$ -transform of $E[-]$. This is possible, since the evaluation strategy ensures, that $E[-]$ does not bind any variables in its subexpression.

To ensure the correctness of the equations below we merely require that $\alpha \notin FV(E^{\mathcal{C}}[-])$. We can safely assume this, since α is bound in $\mu\alpha.s$.

$$[\![E[\mu\alpha.s]]\!]^{\mathcal{C}} = E^{\mathcal{C}}[(\mathcal{C}\ \lambda\alpha.[\![s]\!]^{\mathcal{C}})]$$

$$\begin{aligned} E^{\mathcal{C}}[(\mathcal{C}\ \lambda\alpha.[\![s]\!]^{\mathcal{C}})] &\rightsquigarrow_c (\lambda\alpha.[\![s]\!]^{\mathcal{C}})(\lambda x.(\mathcal{A}\ E^{\mathcal{C}}[x])) \\ &\rightsquigarrow_c [\![s]\!]^{\mathcal{C}}[(\lambda x.(\mathcal{A}\ E^{\mathcal{C}}[x])/\alpha] \\ &= [\![s]\!]^{\mathcal{C}}[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)] \end{aligned}$$

$$E[\mu\alpha.s] \rightsquigarrow_{\mu_c} s[(\mathcal{A}\ E[u])/[\alpha]u]$$

Now we need to show that

$$[\![s[\mathcal{A}(E[u])/[\alpha]u]]\!]^{\mathcal{C}} = [\![s]\!]^{\mathcal{C}}[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)]$$

This can be done by structural induction on the definition of the structural substitution. The only non-trivial case is $s \equiv [\alpha]r$:

$$\begin{aligned}
[\![([\alpha]r)[\mathcal{A}(E[u])/[\alpha]u]]\!]^{\mathcal{C}} &= [\![(\mathcal{A}\ E[r][\mathcal{A}(E[u])/[\alpha]u])]\!]^{\mathcal{C}} \\
&= (\mathcal{A}\ [\![E[r][\mathcal{A}(E[u])/[\alpha]u]]\!]^{\mathcal{C}}) \\
&\qquad\qquad {}^{*}\alpha \notin FV(E^{\mathcal{C}}[-]) \\
&\underset{*}{=} (\mathcal{A}\ E^{\mathcal{C}}[[\![r[\mathcal{A}(E[u])/[\alpha]u]]\!]^{\mathcal{C}}]) \\
\\
[\![[\alpha]r]\!]^{\mathcal{C}}[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)] &= (\alpha\ [\![r]\!]^{\mathcal{C}})[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)] \\
&= (\mathcal{A}\ E^{\mathcal{C}}[[\![r]\!]^{\mathcal{C}}])[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)])
\end{aligned}$$

By the induction hypothesis $[\![r[\mathcal{A}(E[u])/[\alpha]u]]\!]^{\mathcal{C}} = [\![r]\!]^{\mathcal{C}}[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)]$ and therefore $(\mathcal{A}\ [\![r[\mathcal{A}(E[u])/[\alpha]u]]\!]^{\mathcal{C}}) = (\mathcal{A}\ [\![r]\!]^{\mathcal{C}}[(\mathcal{A}\ E^{\mathcal{C}}[w])/(\alpha\ w)])$.

□

Proposition 7.8. *Let t_1 and t_2 be $PROG_K$-terms. If $t_1\ ev_{\overrightarrow{c}}\ t_2$ then $[\![t_1]\!]^{\mu}\ ev_{\overrightarrow{\mu}}\ [\![t_2]\!]^{\mu}$.*

Proof. The proceeding is similar to the proof of proposition 7.7, but shorter (since we do not have to deal with substitution). Again, the result is obvious for the β-rules and the non-trivial case concerns the structural reduction. Therefore we once more keep to the essential cases. We also refrain from doing a case split on the different evaluation contexts but refer to the μ-transform of $[\![E[-]]\!]^{\mu}$ as $E^{\mu}[-]$.

$$\begin{aligned}
[\![E[(\mathcal{C}\ t)]]\!]^{\mu} &= E^{\mu}[\mu\alpha.([\![t]\!]^{\mu}\ \lambda x.[\alpha]x)] \\
&\rightsquigarrow_{\mu_c} ([\![t]\!]^{\mu}\ \lambda x.(\mathcal{A}\ E^{\mu}[x])]
\end{aligned}$$

$$E[(\mathcal{C}\ t)] \rightsquigarrow_c (t\ (\lambda x.(\mathcal{A}\ E[x])))$$

It remains to show that

$$[\![(t\ \lambda x.(\mathcal{A}\ E[x]))]\!]^{\mu} = ([\![t]\!]^{\mu}\ \lambda x.(\mathcal{A}\ E^{\mu}[x])])$$

$$\begin{aligned}
[\![(t\ \lambda x.(\mathcal{A}\ E[x]))]\!]^{\mu} &= ([\![t]\!]^{\mu}\ [\![\lambda x.(\mathcal{A}\ E[x])]\!]^{\mu}) \\
&= ([\![t]\!]^{\mu}\ \lambda x.(\mathcal{A}\ [\![E[x]]\!]^{\mu})) \\
&= ([\![t]\!]^{\mu}\ \lambda x.(\mathcal{A}\ E^{\mu}[x]))
\end{aligned}$$

□

The reduction rules induce an equality relation for each of the two calculi (see chapter 2.5). Using this equality relation on terms, we can also show that the two translations are inverses of each other.

Proposition 7.9. *Let s be a closed $\lambda\mu_c$PRL-term and t be a closed term of $PROG_K$. Then the respective $\mathcal{C}$- and μ-transforms are such that:*

1. $[\![[\![s]\!]^{\mathcal{C}}]\!]^{\mu} =_{\mu} s$

2. $[\![[\![t]\!]^{\mu}]\!]^{\mathcal{C}} =_c t$

Proof. The proofs are done by structural inductions on the respective terms. The non-trivial cases arise from the μ- and $\mathcal{C}$-Operators. We will be content to present only these two:

1. We prove the fact that for any $\lambda\mu_c$PRL-term $[\![[\![s]\!]^{\mathcal{C}}]\!]^{\mu} =_{\mu} s^*$ where s^* is obtained from s by replacing all terms of the form $[\alpha]u$ by $(\alpha\ u)$, i.e. $t^* = t[(\alpha\ u)/[\alpha]u]$.

   The induction hypothesis is $[\![[\![s]\!]^{\mathcal{C}}]\!]^{\mu} =_{\mu} s^*$.

   $$\begin{aligned}
   [\![[\![\mu\alpha.s]\!]^{\mathcal{C}}]\!]^{\mu} &= [\![(\mathcal{C}\ \lambda\alpha.[\![s]\!]^{\mathcal{C}})]\!] \\
   &= \mu\beta.([\![(\lambda\alpha.[\![s]\!]^{\mathcal{C}})]\!]^{\mu}\ (\lambda y.[\beta]y)) \\
   &= \mu\beta.(\lambda\alpha.[\![[\![s]\!]^{\mathcal{C}}]\!]^{\mu}\ (\lambda y.[\beta]y)) \\
   &\qquad\qquad \text{by induction hypothesis} \\
   &\boxed{=_{\mu} \mu\beta.(\lambda\alpha.s^*(\lambda y.[\beta]y))} \\
   &\rightsquigarrow_{\mu_c} (\lambda\alpha.s^*\ (\lambda y.(\mathcal{A}\ y))) \\
   &\rightsquigarrow_{\beta} s^*[\lambda y.(\mathcal{A}\ y)/\alpha] \\
   &\rightsquigarrow_{\beta} s^*[(\mathcal{A}\ w)/(\alpha\ w)] \\
   &= s[(\mathcal{A}\ w)/[\alpha]w)]
   \end{aligned}$$

   $$\mu\alpha.s \qquad \rightsquigarrow_c s[(\mathcal{A}\ w)/[\alpha]w)]$$

   and therefore $[\![[\![\mu\alpha.s]\!]^{\mathcal{C}}]\!]^{\mu} =_{\mu} \mu\alpha.s$.

   Remark To prove that $[\![[\![s]\!]^{\mathcal{C}}]\!]^{\mu} =_{\mu} s^*$ for any $\lambda\mu_c$PRL-term is sufficient, since for closed terms the double-translated term (marked by the box) has the same behaviour in any application-context as the original term. This can easily be verified.

2. The induction hypothesis is $[\![[\![t]\!]^{\mu}]\!]^{\mathcal{C}} =_c t$.

   $$\begin{aligned}
   [\![[\![(\mathcal{C}\ t)]\!]^{\mu}]\!]^{\mathcal{C}} &= [\![\mu\alpha.([\![t]\!]^{\mu}\ \lambda y.[\alpha]y)]\!]^{\mathcal{C}} \\
   &= (\mathcal{C}\lambda\alpha.[\![([\![t]\!]^{\mu}\ \lambda y.[\alpha]y)]\!]^{\mathcal{C}} \\
   &= (\mathcal{C}\ \lambda\alpha.([\![[\![t]\!]^{\mu}]\!]^{\mathcal{C}}\ [\![\lambda y.[\alpha]y]\!]^{\mathcal{C}}) \\
   &\qquad\qquad \text{by induction hypothesis} \\
   &=_c (\mathcal{C}\ \lambda\alpha.(t\ \lambda y.[\![[\alpha]y]\!]^{\mathcal{C}})) \\
   &= (\mathcal{C}\ \lambda\alpha.(t\ \lambda y.(\alpha\ [\![y]\!]^{\mathcal{C}}))) \\
   &= (\mathcal{C}\ \lambda\alpha.(t\ \lambda y.(\alpha\ y))) \\
   &\rightsquigarrow_c (\lambda\alpha.(t\ \lambda y.(\alpha\ y))\ (\lambda x.(\mathcal{A}\ x))) \\
   &\rightsquigarrow_{\beta} (t\ \lambda y.((\lambda x.(\mathcal{A}\ x))\ y))) \\
   &\rightsquigarrow_{\beta} (t\ \lambda y.(\mathcal{A}\ y))
   \end{aligned}$$

   $$(\mathcal{C}\ t) \qquad \rightsquigarrow_c (t\ \lambda y.(\mathcal{A}\ y))$$

   and therefore $[\![[\![(\mathcal{C}\ t)]\!]^{\mu}]\!]^{\mathcal{C}} =_c t$.

□

In conclusion we can establish that $\lambda\mu_c$PRLand $PROG_K$ are isomorphic.

Proposition 7.10. *If t_1 and t_2 are closed terms of $\lambda\mu_c PRL$ and s_1 and s_2 are terms of $PROG_K$, then*

1. $s_1 =_\mu s_2$ *iff* $[\![s_1]\!]^{\mathcal{C}} =_c [\![s_2]\!]^{\mathcal{C}}$
2. $t_1 =_c t_2$ *iff* $[\![t_1]\!]^{\mu} =_\mu [\![t_2]\!]^{\mu}$

Proof. This result can be derived from the propositions 7.7, 7.8 and 7.9. □

7.4 Example Proofs

In this section we will present two example proofs. These were originally given by Murthy in [Mur91b]. Here, we will reproduce them in $\lambda\mu_c$PRL:

- The proof goal in the first example is a decidable Σ_1^0 proposition. Its proof illustrates the inherent non-determinism of classical proofs: the corresponding proof term can be evaluated to two different values, depending on the evaluation strategy.
- The second example concerns the probable unsoundness caused by the combination of the dependent product type, arithmetic and classical reasoning. As in this case an undecidable proposition is proven, the resulting proof terms returns "fake" evidence.

Example 7.1 shows the proof of the first proposition in $\lambda\mu_c$PRL. The resulting term

$$\mu\alpha.[\alpha]\,\langle 102, \langle \mu\beta.[\alpha]\,\langle 2, \pi_2\rangle\,, \mu\gamma.[\alpha]\,\langle 3, \pi_3\rangle\rangle\rangle$$

contains two possible values, but does not prove the incorrect evidence supplied at first (102). Which value is returned by an evaluation procedure depends on the evaluation strategy: Evaluating the term from left to right results in 2, from right to left in 3. This is the reason why either a deterministic evaluator is required or else confluence is lost. π_2 and π_3 denote the respective proofs that for 2 and 3 the proposition really holds.

The second example is a classical proof without constructive content. We assume that $f : \mathbb{N} \to \{0, 1\}$ is a parameter. Then the proof goal $\exists n : \mathbb{N}.\forall m : \mathbb{N}.f(n) \leq f(m)$ expresses the fact that every boolean function will at some point reach a minimum.

If we had a constructive proof we could extract a program for this fact. Now this is intuitively impossible: Consider on the one hand the constant function $f : n \mapsto 1$ and on the other hand a function which constantly returns 1 until it attains its minimum 0 at the input of n_1. The program would not necessarily be able to distinguish between these two functions. The reason is, that for any predetermined finite number N of computation steps, we might have $n_1 > N$. Thus, in case of the second function, the program might return after N steps with a wrong maximum of 1.

The classical proof of the sentence can be found in example 7.2.

$$\mu\alpha.[\alpha]\,\langle 0, \lambda m.if\ f(0) \leq f(m)\ then\ axiom\ else\ \mu\beta.[\alpha]\,\langle m, \lambda m'.axiom\rangle\rangle$$

is the corresponding proof term.

Example 7.1. $\phi \equiv \exists n : \mathbb{N}.prime(n) \wedge n < 100.$

$\vdash \phi$	ext $\mu\alpha.[\alpha]\,\langle 102, \langle \mu\beta.[\alpha]\,\langle 2, \pi_2\rangle\,, \mu\gamma.[\alpha]\,\langle 3, \pi_3\rangle\rangle\rangle$
by mu1 α	
$\vdash \{\{\phi^\alpha\}\} \vdash \bot$	ext $[\alpha]\,\langle 102, \langle \mu\beta.[\alpha]\,\langle 2, \pi_2\rangle\,, \mu\gamma.[\alpha]\,\langle 3, \pi_3\rangle\rangle\rangle$
by abort2 α	
$\vdash \{\{\phi^\alpha\}\} \vdash \phi$	ext $\langle 102, \langle \mu\beta.[\alpha]\,\langle 2, \pi_2\rangle\,, \mu\gamma.[\alpha]\,\langle 3, \pi_3\rangle\rangle\rangle$
by exI 102	
$\vdash \{\{\phi^\alpha\}\} \vdash prime(102) \wedge 102 < 100$	ext $\langle \mu\beta.[\alpha]\,\langle 2, \pi_2\rangle\,, \mu\gamma.[\alpha]\,\langle 3, \pi_3\rangle\rangle$
by andI	
$\vdash \{\{\phi^\alpha\}\} \vdash prime(102)$	ext $\mu\beta.[\alpha]\,\langle 2, \pi_2\rangle$
by mu1 β	
$\vdash \{\{\phi^\alpha\}\}, \{\{prime(102)^\beta\}\} \vdash \bot$	ext $[\alpha]\,\langle 2, \pi_2\rangle$
by abort2 α	
$\vdash \{\{\phi^\alpha\}\}, \{\{prime(102)^\beta\}\} \vdash \phi$	ext $\langle 2, \pi_2\rangle$
by exI 2	
$\vdash \ldots$	ext π_2
by ...	
$\vdash \{\{\phi^\alpha\}\} \vdash 102 < 100$	ext $\mu\gamma.[\alpha]\,\langle 3, \pi_3\rangle$
by mu1 γ	
$\vdash \{\{\phi^\alpha\}\}, \{\{102 < 100^\gamma\}\} \vdash \bot$	ext $[\alpha]\,\langle 3, \pi_3\rangle$
by abort2 α	
$\vdash \{\{\phi^\alpha\}\}, \{\{102 < 100^\gamma\}\} \vdash \phi$	ext $\langle 3, \pi_3\rangle$
by exI 3	
$\vdash \ldots$	ext π_3
by ...	

Example 7.2. $\psi \equiv \exists n : \mathbb{N}.\forall m : \mathbb{N}.f(n) \leq f(m)$
$\xi \equiv f(0) \leq f(m)$

$\vdash \psi$	ext $\mu\alpha.[\alpha]\,\langle 0, \lambda m.if\ \ldots\rangle$
by mu1 α	
$\vdash \{\{\psi^\alpha\}\} \vdash \bot$	ext $[\alpha]\,\langle 0, \lambda m.if\ \ldots\rangle$
by abort2 α	
$\vdash \{\{\psi^\alpha\}\} \vdash \psi$	ext $\langle 0, \lambda m.if\ \ldots\rangle$
by exI 0	
$\vdash \{\{\psi^\alpha\}\} \vdash \forall m : \mathbb{N}.f(0) \leq f(m)$	ext $\lambda m.if\ f(0) \leq f(m)\ \ldots$
by allI m	
$\vdash m : \mathbb{N}, \{\{\psi^\alpha\}\} \vdash f(0) \leq f(m)$	ext $if\ f(0) \leq f(m)\ then\ axiom$
by cases on $f(0) \leq f(m)$	$else\ \mu\beta.[\alpha]\,\langle m, \lambda m'.axiom\rangle$
$\vdash m : \mathbb{N}, f(0) \leq f(m), \{\{\psi^\alpha\}\} \vdash f(0) \leq f(m)$	ext $axiom$
by hypothesis	
$\vdash m : \mathbb{N}, f(m) < f(0), \{\{\psi^\alpha\}\} \vdash f(0) \leq f(m)$	ext $\mu\beta.[\alpha]\,\langle m, \lambda m'.axiom\rangle$
by mu2 β	
$\vdash m : \mathbb{N}, f(m) < f(0), \{\{f(0) \leq f(m)^\beta\}\}, \{\{\psi^\alpha\}\} \vdash \bot$	ext $[\alpha]\,\langle m, \lambda m'.axiom\rangle$
by abort2 α	
$\vdash m : \mathbb{N}, f(m) < f(0), \{\{\xi^\beta\}\}, \{\{\psi^\alpha\}\} \vdash \psi$	ext $\langle m, \lambda m'.axiom\rangle$
by exI m	
$\vdash m : \mathbb{N}, f(m) < f(0), \{\{\xi^\beta\}\}, \{\{\psi^\alpha\}\} \vdash \forall m' : \mathbb{N}.f(m) \leq f(m')$	ext $\lambda m'.axiom$
by allI m'	
$\vdash m : \mathbb{N}, f(m) < f(0), \{\{\xi^\beta\}\}, \{\{\psi^\alpha\}\}, m' : \mathbb{N} \vdash f(m) \leq f(m')$	ext $axiom$
by ...	
$\vdash \ldots$	

ext $\mu\alpha.[\alpha]\,\langle 0, \lambda m.if\ f(0) \leq f(m)\ then\ axiom\ else\ \mu\beta.[\alpha]\,\langle m, \lambda m'.axiom\rangle\rangle$

Obviously, the evidence extracted from the proof is no real evidence. Instead, we obtain a program which at first arbitrarily chooses 0 as n. When given m, it checks whether the condition $f(0) \leq f(m)$ is fulfilled. Now it either reports success (if the statement was true), otherwise it assumes $f(m) < f(0)$. In this case $f(m)$ must be 0. So the program jumps back to before 0 was chosen and uses m instead.

Murthy summarizes the behaviour of this program as follows:

> "..., our program does not really provide evidence for the truth of the proposition it purports to be a proof of, but rather, provides a program which, given a counterexample, will "throw" back to a place in the computation where it can change the "answer" to disqualify the counterexample."[Mur91b]

However, this does not pose a problem for the soundness of the calculus: The example is out of the original range of $\lambda\mu_c$PRL which is restricted to decidable propositions. Otherwise the example would render the calculus unsound.

Remark The term `if` b `then` s `else` t is used in the proof of example 7.2 for brevity of presentation. It is short for `case` b `of inl(`x`):` s `| inr(`y`):` t, where b is either `inl(`$axiom$`)` or `inr(`$axiom$`)`. These two are used to define the boolean values `true` and `false` in Nuprl (as conservative extension, see chapter 2.2).

Still, it is interesting to see what happens when proving undecidable propositions. The question is, whether it is possible to identify "fake" evidence in a proof term. Then we could again leave the decision to the user (as intended with the propositional $\lambda\mu$PRL).

Possibly this could be achieved by adapting the concept of μ-safety from chapter 6 to $\lambda\mu_c$PRL. We will not pursue this in detail here, but just have a short look at the proof term of example 7.2: If we ignore the case split contained in the term (as *axiom* is not part of the definition of μ-safety), it is still quite obvious that this term cannot be μ-safe anyway. The reason is the λ-variable m which is free in the scope of the μ-variable α.

We conjecture that the concept of μ-safety could be employed to distinguish real and fake evidence in $\lambda\mu_c$PRL-terms.

Chapter 8

Conclusion and Perspectives

In this chapter we consider the results of this thesis and possible directions for future work.

8.1 Conclusion

The original goal of this thesis was to develop a calculus for classical reasoning within computational type theory [Con08]. This calculus was intended to enable users to directly employ classical logic in a constructive environment. At the same time the use of classical logic was supposed to be recognizable in retrospect.

To achieve this aim, we have provided $\lambda\mu$PRL, a sequent-style proof refinement calculus for classical propositional logic. $\lambda\mu$PRL is based on a variant of Parigot's $\lambda\mu$-calculus [Par92] and specifically targets the computational type theory of the Nuprl proof development system [CAB$^+$86]. Therefore a large part of its properties were influenced by $\lambda\mu$ and Nuprl's logic and operational semantics.

Having started from the multi-conclusioned $\lambda\mu\nu$-calculus by Pym and Ritter, we first considered different possibilities of integrating this calculus into the Nuprl system. The reasons to settle on a single-conclusioned variant have been presented in chapter 3.

These considerations have led to the calculus $\lambda\mu$PRL. The calculus and a soundness proof have been given in chapter 4. In hindsight, it might have been sensible to base this proof on de Groote's full propositional $\lambda\mu$-calculus [dG01]. This would have been more straightforward, since de Groote's calculus already uses the constructive disjunction. Moreover it might have been possible to preserve (some of) the calculus' proof-theoretic features by a translation.

The decision to use Pym's and Ritter's $\lambda\mu\nu$-calculus resulted from the origins of this work. These included the considerations, as to whether a multi-conclusion was suited for an integration of classical reasoning into Nuprl. Therefore it was also the first choice to derive the soundness of $\lambda\mu$PRL from $\lambda\mu\nu$.

In chapter 5 we defined an accompanying reduction relation in the context of reduction in different $\lambda\mu$ calculi. In the scope of this work we were content with $\beta\mu$-reduction, which is shown to satisfy the confluence property. Besides we proposed reduction rules for permutative conversions. These might provide the reduction relation with desirable proof-theoretic properties like strong normalization. However, we have not pursued this matter any further.

$\lambda\mu$PRL shares the rules for implication, conjunction and disjunction and the axiom-rule with Nuprl's logic. Additionally, there are two more rules which are identified with the special terms μ and $[-]$. These rulesextend the otherwise constructive calculus to classical logic. Therefore the occurrence of the terms μ and $[-]$ in a term can be used to identify classical reasoning in a proof.

Nevertheless, not every term containing these constructs represents a non-constructive proof. The possibility to make finer distinctions between non-constructive and constructive terms has been shown in chapter 6. For this purpose we adopted a notion of *μ-safe* terms, originally introduced by Crolard [Cro02]. Using this concept, we can identify whether the crucial hypotheses of a $\lambda\mu$PRL-proof are also available in a purely intuitionistic proof.

Still, there are proofs in $\lambda\mu$PRL which are both constructive and non-constructive. This is due to a generic non-determinism of classical logic. Thus it depends on the evaluation strategy whether the proof term in question produces a value or not. To treat such terms, we proposed the notion of *partial μ-safety*.

Finally, we considered the relation between $\lambda\mu$PRLand $PROG_K$, a calculus from an early work on computational content of classical proofs by Murthy [Mur91b]. $PROG_K$ is also related to Nuprl's type theory and extends it to classical logic (for Σ^0_1 and Π^0_2 sentences). Therefore it shares the rules for implication, conjunction and disjunction with $\lambda\mu$PRL. Clarifying the relation between $\lambda\mu$PRL and $PROG_K$ induces a variant of $\lambda\mu$PRL for Peano Arithmetic.

This variant $\lambda\mu_c$PRL is defined in chapter 7. We provide a translation between $\lambda\mu_c$PRL and $PROG_K$ and show that by adapting the evaluation rules of $PROG_K$ to $\lambda\mu_c$PRL, we can establish an isomorphism between the two calculi. As a consequence it is possible to extract constructive content from $\lambda\mu_c$PRLproofs of decidable formulae, using the means of $PROG_K$.

As described in this section, we have developed $\lambda\mu$PRL and examined its features. In this context, we considered proof-theoretic properties, the possibility of distinguishing classical from constructive proofs, and the relation to earlier work in this area. Unsurprisingly, each of these aspects has risen a number of further questions. In the next section we will address some of these.

8.2 Perspectives

There are various possible directions for future work. In this section, we will try to address some of these.

From a practical point of view, the next step might be to integrate $\lambda\mu$PRL into the Nuprl system. The extended system could be used to develop larger exemplary proofs.

As we closely oriented our design decisions on the system, a basic implementation could be done quite straightforwardly. All rules except the rules `mu` and `abort` already exist in Nuprl's inference system and the necessary new constructs could be easily created using Nuprl's meta-language. Proofs containing one of the new constructs in their proof term could be marked by some label to ensure the soundness of the system and more elaborate means to distinguish constructive from non-constructive proofs could be added later. Besides, the proof term of Nuprl proofs is computed after the completion of a proof. A proof is not accepted as valid before this check. As a consequence, this procedure might be well suited to also include further evaluation of the proof term.

Such further evaluation is another possibility for future work. With the concept of μ-safety, there is a basis for "finer" distinctions between constructive and non-constructive proofs. But this does not cover all terms containing constructive content. Thus it would be interesting to extend the approach of partial μ-safety. Then a sequence of open questions follows up:

- Can a decision procedure for partial μ-safety identify every proof term with constructive content?
- If a term is not even partially μ-safe, does it contain any information on the possibilty of proving the same formula constructively?
- Does the existence of constructive content in a proof term imply the decidability of the original proposition?
- Can the concept of μ-safety and partial μ-safety be extended to $\lambda\mu_c$PRL?
- Do partially μ-safe $\lambda\mu_c$PRL-terms produce a value when evaluated appropriately?
- If so, could $\lambda\mu_c$PRL be used without restriction on its goal formulae? Would it be possible to subsequently identify non-decidable sentences and thereby re-establish soundness?

An extension of the concept to $\lambda\mu_c$PRL is also interesting for a general extension of $\lambda\mu$PRL to first order logic and arithmetic. The problem with an extension of $\lambda\mu$PRL without restricting it to decidable formulae, is related to the axiom of choice. In Nuprl, the existential quantifier is defined by the dependent product type (sometimes also referred to as the strong sum type) via the propositions-as-types principle. When used for program synthesis, the proof of a specification therefore returns a pair $\langle e, p\rangle$ which consists of the algorithm and the proof term guaranteeing its correctness.

However, in presence of the classical rules of $\lambda\mu$PRL, the proof corresponding to p might not really prove the correctness of e. Instead it may dismiss the original evidence at some point and prove something else.

In fact, it can be proved that an arithmetic type theory which contains the dependent product type and validates the principle of the excluded middle is unsound. A proof of this is given by Stewart [Ste99] (who follows an argument of Coquand [Coq95]). Another illustration of the problem, focussing on the axiom of choice, can be found in [Mur90]. The basic idea is that the non-decidable halting problem could be shown to be decidable in such a theory.

Still, there might be methods to circumvent this problem. In [How96a, How96b], Howe provides an operational semantics for Nuprl which allows an import of results from HOL. But since the HOL proofs can contain instances of HOL's non-constructive "select"-operator, Howe makes sure these results cannot be used when computation is required. This is achieved by marking the proof's root node whenever it is not computational.

As we can see, Howe makes use of Nuprl's meta-system to ensure soundness. Now, if it were possible to identify non-constructive $\lambda\mu_c$PRL-proofs, we could use a similar method. This is where the concept of partial μ-safety might be applicable.

Another question arising in the context of Howe's work concerns the relation between the μ-operator of $\lambda\mu$PRL and HOL's select-operator. This also leads to the subject of

the computational meaning of seemingly non-constructive $\lambda\mu$PRL- (respectively $\lambda\mu_c$PRL-) terms. It is well-known, that the μ-operator can be seen as "generic jump operator" [OS97]. Therefore the terms of $\lambda\mu$PRL correspond to functional programming languages with control operators.

Ong exploits this correspondence to define μPCF, a classical extension of PCF [Sco93]. In this context, he defines typing rules for different kinds of control constructs: the `Y`-combinator, Scheme's `call/cc` and ML-style exception handling via `throw` and `catch`. Similar approaches have been taken by other authors as well. E.g. in [dG95] de Groote presents a calculus for exception handling which is closely related to $\lambda\mu$.

Alongside these results, an extension of Nuprl's purely functional programming language might be of interest as well. It should be considered, whether on this foundation a semantics of evidence for a hybrid type theory could be developed.

List of Figures

2.1 Types in Nuprl 6
2.2 Conservative extensions in Nuprl 6
2.3 $PROG_K$-terms 7
2.4 $PROG_K$-reduction-rules 8
2.5 The inference rules of $\lambda\mu\nu$ 11

4.1 The calculus $\lambda\mu$PRL 26
4.2 The calculus $\lambda\mu\nu_\perp$ 27
4.3 The calculus $\lambda\mu\oplus$ 27
4.4 The calculus $\lambda\mu\oplus_\perp$ 27

6.1 The inference rules of $\lambda\mu_r$PRL 56

7.1 The calculus $\lambda\mu_c$PRL for Peano Arithmetic 64

Bibliography

[Acz78] Peter Aczel. A general church-rosser theorem. Technical report, University of Manchester, July 1978. 41

[AH08] Zena M. Ariola and Hugo Herbelin. Control reduction theories: the benefit of structural substitution. *J. Funct. Program.*, 18(3):373–419, 2008. 39

[AHS07] Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A type-theoretic foundation of delimited continuations. *Higher Order and Symbolic Computation*, 2007. 37

[Bar84] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics, Revised edition.* North Holland, 1984. 14, 15, 16, 41

[BB92] Franco Barbanera and Stefano Berardi. A constructive valuation interpretation for classical logic and its use in witness extraction. In J.-C. Raoult, editor, *Proceedings of Colloquium on Trees in Algebra and programming (CAAP '92)*, volume 581 of *Lecture Notes in Computer Science*, pages 1–23. Springer Verlag, 1992. 2

[BB93] Franco Barbanera and Stefano Berardi. Extracting constructive content from classical logic via control-like reductions. In *In Bezem and Groote*, pages 45–59. Springer-Verlag, 1993. 2, 37

[BBS95] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Program extraction from classical proofs. In *Annals of Pure and Applied Logic*, pages 77–97. Springer Verlag, 1995. 2

[BHF01] Kensuke Baba, Sachio Hirokawa, and Ken-etsu Fujita. Parallel reduction in type free $\lambda\mu$-Calculus. *Electr. Notes Theor. Comput. Sci.*, 42, 2001. 2, 9, 37, 39, 41, 43

[Böh68] Corrado Böhm. Alcune proprietà delle forme normali nel **K**-calcolo. Technical Report 696, INAC, Roma, Italy, 1968. 38

[Bre08] Nuria Brede. Klassisches Schliessen in der intuitionistischen Typentheorie: Entwicklung eines Typs zur Simulation von Multikonklusionen. Student project, Department of Computer Science, University of Potsdam, 2008. 2, 18

[CAB+86] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler,

P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ, 1986. 1, 5, 73

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940. 1

[Con98] R. L. Constable. Types in logic, mathematics, and programming. In Samuel R. Buss, editor, *Handbook of Proof Theory*, pages 683–786. Elsevier Science Publishers, Amsterdam, 1998. 1, 5

[Con08] Robert L. Constable. Computational type theory. Technical Report 1813/11512, Cornell University, October 2008. 1, 73

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *stoc71*, pages 151–158, 1971. 1, 5

[Coq95] Thierry Coquand. A semantics of evidence for classical arithmetic. *J. Symb. Log.*, 60(1):325–337, 1995. 2, 75

[Cro02] Tristan Crolard. A constructive restriction of the $\lambda\mu$-calculus. Technical Report 02, UFR d'Informatique, Université Paris 7, 2002. 1, 3, 51, 54, 74

[dG94a] Philippe de Groote. A cps-translation of the $\lambda\mu$-calculus. In Sophie Tison, editor, *Trees in Algebra and Programming – CAAP94, 19th International Colloquium*, volume 787 of *Lecture Notes in Computer Science*, pages 85–99, Edinburgh, 1994. Springer Verlag. 37

[dG94b] Philippe de Groote. On the relation between the lambda-mu-calculus and the syntactic theory of sequential control. In *LPAR*, pages 31–43, 1994. 39, 62, 63

[dG95] Philippe de Groote. A simple calculus of exception handling. In *TLCA*, pages 201–215, 1995. 2, 76

[dG01] Philippe de Groote. Strong normalization of classical natural deduction with disjunction. In *TLCA*, pages 182–196, 2001. 2, 37, 39, 73

[DN03] Rene David and Karim Nour. A short proof of the strong normalization of classical natural deduction with disjunction. *Journal of Symbolic Logic*, 68(4):1277–1288, 2003. 39

[DP01] René David and Walter Py. lambda mu - calculus and böhm's theorem. *J. Symb. Log.*, 66(1):407–413, 2001. 38

[FFKD87] Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. A syntactic theory of sequential control. *Theor. Comput. Sci.*, 52:205–237, 1987. 1, 7, 48, 62

[FH92] Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992. 1, 7, 48, 62

[Fri78] Harvey M. Friedman. Classically and intuitionistically provably recursive functions. In *Higher Set Theory*, volume 669 of *Lecture Notes in Mathematics*, pages 21–27. Springer-Verlag, 1978. 1, 7

[Fuj97] Ken-etsu Fujita. Calculus of classical proofs i. In *ASIAN*, pages 321–335, 1997. 2, 37

[Gal91] Jean H. Gallier. Constructive logics, part I: A tutorial on proof systems and typed λ-calculi. Research Report 8, DEC Paris Research Laboratory, Reuil-Malmaison, may 1991. 1, 5

[Gen34] Gerhard Gentzen. Untersuchungen über das logische schliessen. *Mathematische Zeitschrift*, 39:176–210, 1934. 4, 11, 17

[Gha95] Neil Ghani. fij-equality for coproducts. In *In Typed -calculus and Applications, number 902 in Lecture Notes in Computer Science*, pages 171–185. Springer Verlag, 1995. 39

[Gir91] Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991. 2

[GM93] Michael J. C. Gordon and Thomas F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993. HOL4 website: http://hol.sourceforge.net. 1

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In *POPL*, pages 47–58, 1990. 1, 7, 9, 61

[Hey71] A. Heyting. *Intuitionism: An Introduction*. North-Holland, Amsterdam, 3 edition, 1971. 7

[Hin64] J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle-upon-Tyne, 1964. 40

[How80] William A. Howard. The formulae-as-type notion of construction, 1969. In J. P. Seldin and R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, New York, 1980. 1, 7, 24

[How96a] Douglas J. Howe. Importing Mathematics from HOL into Nuprl. In *Theorem Proving in Higher Order Logics, 9th International Conference*, pages 267–281, 1996. 2, 75

[How96b] Douglas J. Howe. Semantic Foundations for Embedding HOL in Nuprl. In *Algebraic Methodology and Software Technology, 5th International Conference*, pages 85–101, 1996. 2, 75

[Klo92] J.W. Klop. *Handbook of Logic in Computer Science*, volume 2, chapter Term rewriting systems, pages 1–116. Oxford University Press, 1992. 16

[Kre02] Christoph Kreitz. *The Nuprl Proof Development System, Version 5, Reference Manual and User's Guide*. Cornell University, 2002. 5

[Kre04] Christoph Kreitz. Building reliable, high-performance networks with the nuprl proof development system. *J. Funct. Program*, 14(1):21–68, 2004. 5, 7

[Mat01] Ralph Matthes. Parigots second order $\lambda\mu$-calculus and inductive types. In S. Abramsky, editor, *Proceedings of TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 329–343. Springer-Verlag Berlin Heidelberg, 2001. 2, 9, 37, 39

[Men88] Nax P. Mendler. *Inductive Definition in Type Theory.* PhD thesis, Cornell University, 1988. 3

[ML84] Per Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory: Lecture Notes.* Bibliopolis, Napoli, 1984. 5

[Mur90] Chetan R. Murthy. *Extracting Constructive Content from Classical Proofs.* PhD thesis, Department of Computer Science, Cornell University, August 1990. 2, 7, 8, 9, 39, 58, 75

[Mur91a] Chetan R. Murthy. Classical proofs as programs: How, what, and why. In *Constructivity in Computer Science*, pages 71–88, 1991. 2, 7, 9

[Mur91b] Chetan R. Murthy. An evaluation semantics for classical proofs. In *LICS*, pages 96–107, 1991. 1, 2, 3, 7, 23, 48, 61, 70, 72, 74

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Math.*, 2(43):223–243, 1942. 41

[NS06] Karim Nour and Khelifa Saber. A semantical proof of the strong normalization theorem for full propositional classical natural deduction. *Arch. Math. Log.*, 45(3):357–364, 2006. 37, 39

[NT03] Koji Nakazawa and Makoto Tatsuta. Strong normalization proof with cps-translation for second order classical natural deduction. *J. Symb. Log.*, 68(3):851–859, 2003. 37

[NT08] Koji Nakazawa and Makoto Tatsuta. Strong normalization of classical natural deduction with disjunctions. *Annals of Pure and Applied Logic*, 153:21–37, 2008. 37, 39

[Oga98] Ichiro Ogata. Classical proofs as programs, cut elimination as computation. Technical report, Electrotechnical Laboratory, 1998. 2

[Ong96] C.-H. L. Ong. A semantic view of classical proofs. In *In Proceedings of LICS '96.* IEEE Press, 1996. 3, 10, 24

[OS97] C.-H. L. Ong and C. A. Stewart. A curry-howard foundation for functional computation with control. In *In Proceedings of ACM SIGPLAN-SIGACT Symposium on Principle of Programming Languages*, pages 215–227. ACM Press, 1997. 2, 3, 5, 10, 24, 39, 76

[Par92] Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR*, pages 190–201, 1992. 2, 9, 37, 38, 41, 61, 73

[Par93a] Michel Parigot. Classical proofs as programs. In *Computational logic and proof theory*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, 1993. 2, 9, 39, 47

[Par93b] Michel Parigot. Strong normalization for second order classical natural deduction. In *LICS*, pages 39–46, 1993. 37

[PdP05] Luiz Carlos Pereira and Valeria de Paiva. A short note on intuitionistic propositional logic with multiple conclusions. *MANUSCRITO - Rev. Int. Fil.*, 28(2):317–329, jul-dez 2005. 51

[PR01] David J. Pym and Eike Ritter. On the semantics of classical disjunction. *Journal of Pure and Applied Algebra*, 159:315–338, 2001. 2, 10, 24, 25, 38

[PR04] David J. Pym and Eike Ritter. *Reductive Logic and Proof-search*. Oxford University Press, 2004. 1, 2, 10, 24, 25

[Py98] Walter Py. *Confluence en $\lambda\mu$-calcul*. PhD thesis, UFR Sciences Fondamentales et Appliquées, Université de Savoie, 1998. 2, 9, 14, 37, 39, 40, 41

[Pym] David J. Pym. Notes towards a semantics for proof-search. 2, 10

[Ros73] Barry K. Rosen. Tree-manipulating systems and church-rosser theorems. *J. ACM*, 20(1):160–187, 1973. 40

[RPW00] Eike Ritter, David Pym, and Lincoln Wallen. On the intuitionistic force of classical search. *Theoretical Computer Science*, 232(1–2):299–333, 2000. 2, 10

[RS94] Niels Jakob Rehof and Morten Heine Sørensen. The λ_Δ calculus. In *Theoretical Aspects of Computer Software*, pages 516–542. Springer-Verlag, 1994. 2, 37

[Sco93] Dana S. Scott. A type-theoretical alternative to iswim, cuch, owhy. *Theor. Comput. Sci.*, 121(1&2):411–440, 1993. 76

[Sta79] Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci.*, 9:67–72, 1979. 1, 5

[Ste99] Charles A. Stewart. *On the formulae-as-types correspondence for classical logic*. PhD thesis, Worcester College, Oxford University, 1999. 2, 3, 24, 75

[Tat07] Makoto Tatsuta. The maximum length of mu-reduction in lambda mu-calculus. In *RTA*, pages 359–373, 2007. 37

[UB01] Christian Urban and Gavin M. Bierman. Strong normalisation of cut-elimination in classical logic. *Fundam. Inform.*, 45(1–2):123–155, 2001. 51

[Yam04] Yoriyuki Yamagata. Strong normalization of the second order symmetric $\lambda\mu$ calculus. In *Information and Computation*, volume 193. Elsevier, 2004. 37

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass ich die von mir eingereichte Diplomarbeit bzw. die von mir namentlich gekennzeichneten Teile selbständig verfasst und ausschließlich die angegebenen Hilfsmittel benutzt habe.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Potsdam, den 27. Januar 2009

(Nuria Brede)