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

Nuria Brede and Christoph Kreitz

Institut für Informatik, Universität Potsdam,
August-Bebel-Strasse 89, 14482 Potsdam, Germany
{brede,kreitz}@cs.uni-potsdam.de

Abstract. We present a hybrid proof calculus $\lambda\mu$PRL that combines the propositional fragment of computational type theory with classical reasoning rules from the $\lambda\mu$-calculi. The calculus supports the top-down development of proofs as well as the extraction of proof terms in a functional programming language extended by a nonconstructive binding operator. It enables a user to employ a mix of constructive and classical reasoning techniques and to extract algorithms from proofs of specification theorems that are fully executable if classical arguments occur only in proof parts related to the validation of the algorithm.

We prove the calculus sound and complete for classical propositional logic, introduce the concept of μ-safe terms to identify proof terms corresponding to constructive proofs and show that the restriction of $\lambda\mu$PRL to μ-safe proof terms is sound and complete for intuitionistic propositional logic. We also show that an extension of $\lambda\mu$PRL to arithmetical and first-order expressions is isomorphic to Murthy's calculus $PROG_K$.

Key words: Computational type theory, classical logic, $\lambda\mu$-calculi

1 Introduction

Computational Type Theory [1,5] is a constructive higher order logic that supports the extraction of functional programs from proofs of specification theorems and guarantees that these programs are *correct by construction*. The ability to synthesize programs via the *proofs-as-programs* correspondence is founded on a *constructive notion of existence*. This notion is the fundamental difference between intuitionistic (constructive) and classical logic. While in intuitionistic logic objects can only be proven to exist by showing how to construct them, classical logic allows proofs by contradiction and the existence of objects is not necessarily tied to actually knowing them.

Since intuitionistic logic rejects classical axioms like the law of the excluded middle, proof search in intuitionistic logic is more difficult than in classical logic. Whereas proof search in classical propositional logic is co-$\mathcal{NP}$-complete, it is $\mathcal{PSPACE}$-complete in intuitionistic logic [7,31].

It was believed that classical logic had no constructive content until Griffin [16] proposed an extension of the proofs-as-programs-principle to classical logic. He combined the simply typed λ-calculus with Felleisen's control operator $\mathcal{C}$ [13] which he typed with the classical axiom of double negation elimination. This relates classical proofs to functional programming languages with access to imperative control-flow constructs such as the exception handling mechanism in ML or the `call/cc`- operator in Scheme.

Griffin's work triggered a lot of research on the semantics and constructive content of classical proofs, among these Murthy's [21,22] concerning the constructive content of Σ_1^0 sentences in Peano arithmetic. Murthy's work was situated in the context of the Nuprl system [6] and contributed among other results a method for obtaining values directly from classical proof terms.

Parigot [26] investigated an algorithmic interpretation of natural deduction by designing a classical calculus $\lambda\mu$, which uses a new binding operator μ instead of the additional operator $\mathcal{C}$. The μ-operator binds a new kind of variables that indicate applications of classical reasoning in the sense of switching the goal formula or exchanging the control context. On this basis multiple variants of $\lambda\mu$-calculi have been studied by a variety of authors (e.g. [25,9,12,29]).

In this paper we aim at creating a foundation for a hybrid proof system that integrates $\lambda\mu$ into Computational Type Theory and enables users to decide which logic they want to employ in different parts of a proof and thus to benefit from the advantages of both concepts. A crucial aspect of this investigation is to find a way of maintaining the system's soundness in presence of both constructive and classical reasoning.

While Howe's embedding of HOL (classical) [15] into Nuprl (constructive) [18] has demonstrated that the semantics of a computational type theory can be reconciled with classical logic there are also limitations. Systems that include both classical axioms and a constructive notion of existence become unsound in presence of arithmetic (see e.g. [21]). Howe solved this problem by ensuring that HOL proofs (which do not have any algorithmic content) may not be used within Nuprl proof fragments that require a construction. To implement this condition he added a parameter to proof nodes in Nuprl, which indicates whether the respective proof may be used for construction or not.

Our research follows a similar line of thought: if the use of classical reasoning was visible in the proof term it would enable us to subsequently distinguish between constructive and non-constructive proofs. To examine this approach in a contained range, we have developed a propositional top-down calculus $\lambda\mu$PRL that combines the propositional rules of Computational Type Theory with the classical exchange rules that are characteristic for $\lambda\mu$-calculi.

In the following section we will briefly review Computational Type Theory and $\lambda\mu$-calculi and then present the calculus $\lambda\mu$PRL in section 3. We examine how constructive and classical reasoning is reflected in the extracted proof terms in section 4. In this context we adapt Crolard's notion of *μ-safe* [9] and propose an extension of this concept to *partially μ-safe* terms. In section 5 we investigate the relation between $\lambda\mu$PRL and Murthy's work, showing how constructive content can be extracted from proofs in a variant of $\lambda\mu$PRL. Finally we discuss possibilities for future investigations in section 6. Proof details that cannot be presented here can be found in [4].

	Type	Members and associated non-canonical expressions
Function Space	$S \to T, x : S \to T$	$\lambda x.t, f\ t$
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`)` $\mapsto v$
Logical connectives	$\forall\ \exists \wedge \vee \Rightarrow \neg\ True\ False$	*— Curry-Howard isomorphism —*

Fig. 1. CTT types relevant for propositional reasoning

2 Preliminaries

2.1 Computational Type Theory

Computational Type Theory (CTT) [1] is an extension of Martin-Löf's intuitionistic type theory [20], designed as an open-ended foundation for reasoning about computation in both mathematics and computing practice. It is also the logic of the Nuprl proof development system [6,19], which supports interactive and tactic-based proof development and provides a tool for the synthesis, verification, and optimization of programs.

In CTT types are associated with sets of rules about the formation, decomposition, and extensional equality of types and their members as well as the evaluation of non-canonical expressions. Within this paper we focus on the fragment of CTT types relevant for propositional reasoning, which is summarized in figure 1.

CTT-expressions are defined independently of their type. Thus even constructs like the Y-combinator may be used in terms. This makes it possible to represent all computable functions as terms even if they cannot be typed. In a proof, however, one has to show that an expression is a member of some type. To maintain termination of evaluation in presence of such expressiveness CTT employs a lazy evaluation strategy.

Proofs 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 the system's library. In Nuprl, this library contains a wide range of formalized mathematical knowledge.

Proof goals can be manipulated by user-definable proof tactics. Users may apply tactics that execute a single inference rule, combine existing tactics through the use of predefined *tacticals*, or write arbitrary expression of the system's meta-language (an early member of the ML family of programming languages) that analyze the proof goal to decide which series of inference rules should be applied.

Users may also introduce conservative extensions of CTT via *abstract definitions*. With this mechanism new types inherit the properties of the types used in their definition. An example of such an extension is the embedding of constructive logic into CTT via the propositions-as-types-principle, also known as the Curry-Howard-isomorphism [17]. The definition of conjunction is based on the product type, disjunction on the sum type, implication on the function type, existential and universal quantification on dependent product and function (i.e. Σ- and Π-) types. Classical logic can then be used via the Gödel translation.

2.2 $\lambda\mu$-calculi

The $\lambda\mu$-calculus was first introduced by Parigot [26,27] as an algorithmic interpretation of classical natural deduction well suited for proof theoretic studies. The original version of $\lambda\mu$ is a multi-conclusioned second order calculus. μ-variables "name" the formulae of the multi-conclusion and the μ-operator marks the application of structural rules like exchanging the active formula in the succedent. Parigot's $\lambda\mu$ has a confluent cut elimination procedure and is strongly normalizable [26,27,28,2].

Numerous variants of $\lambda\mu$-calculi have been presented. One of these is a $\lambda\mu$ type theory by Ong and Stewart [25,24], which returns to a single conclusion and keeps the non-active μ-formulae in the antecedent as μ-context. Another variant is Pym's and Ritter's multi-conclusioned $\lambda\mu\nu$-calculus [29] for full propositional logic 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.

3 The Calculus $\lambda\mu$PRL

$\lambda\mu$PRL is based on the logical rules of CTT associated with the types on which the logical connectives are defined. Similar to Ong and Stewart's calculus [24,25] it keeps passive formulae of the μ-context on the left-hand side of a sequent and distinguishes them from the λ-context by their syntactic form. Their role is equivalent to the non-active formulae of Parigot's $\lambda\mu$-multi-conclusion since they are separated from the rest of the sequent by their names (μ-variables) and accessible only by the specific exchange rules. The rules of $\lambda\mu$PRL are presented in figure 2.

Instead of allowing an empty conclusion after the application of the μ-rule and before the application of the $[-]$-rule we use logical absurdity $\bot$, which in CTT is defined via the empty data type `void`, thus expressing the "emptiness" of the active position. Additional rules for the introduction or elimination of $\bot$ are not required. This approach also coincides with Ong's $\lambda\mu$-theory.

For the treatment of disjunction $\lambda\mu$PRL uses the "constructive" $\vee$ rules corresponding to CTT's sum type (which is similar to de Groote's $\lambda\mu$ calculus with disjunction [12] or Pym's and Ritter's $\lambda\mu\oplus$ [30]). 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 in the μ-context. Some of the rules need additional parameters:

- The elimination- and `abort`-rules as well as the `hypothesis`-rule need the position of the formula to operate on denoted by $\$i$.
- The `mu`-rules and `lambdaI` need the name of a new variable to be declared.

Types range over $A, B, C, \ldots$ and are built by the constructors $\supset, \wedge, \vee$ from a set of atomic types and the distinguished absurdity type $\bot$. The negation $\neg A$ is expressed as $A \supset \bot$. Terms of $\lambda\mu$PRL are built by the grammar in figure 2. Like CTT-terms, $\lambda\mu$PRL terms are untyped but have to be associated with types in proofs.

$\lambda\mu$PRL-sequents can be viewed as term assignment judgments. A sequent of the form $\Gamma \vdash T \ \texttt{ext} \ t$ means that the type T is inhabited by the term t. Γ ranges over both λ- and μ-contexts. Accordingly there are two kinds of variables, λ- and

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

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 orI1
  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

β-reduction:

$(\lambda x.u)\ v$	$\leadsto_\beta u[v/x]$
$spread(\langle v,w\rangle; x,y.u)$	$\leadsto_\beta u[v,w/x,y]$
$decide(inl(w); x.u; y.v)$	$\leadsto_\beta u[w/x]$
$decide(inr(w); x.u; y.v)$	$\leadsto_\beta v[w/y]$
$[\alpha]\mu\beta.u$	$\leadsto_\beta u[\alpha/\beta]$

μ-reduction:

$(\mu\alpha.u\ v)$	$\leadsto_\mu \mu\alpha.u[[\alpha](w\ v)/[\alpha]w]$
$spread(\mu\alpha.z, x, y.u)$	$\leadsto_\mu \mu\alpha.z[[\alpha]spread(w; x,y.u)/[\alpha]w]$
$decide(\mu\alpha.z; x.u; y.v)$	$\leadsto_\mu \mu\alpha.z[[\alpha]decide(w; x.u; y.v)/[\alpha]w]$

Fig. 2. The calculus $\lambda\mu$PRL

μ-variables, ranging over two distinct alphabets. In a sequent λ-variables are declared by $x : A, y : B, \ldots$ and bound by the λ-abstraction $\lambda x.t$. Similarly, μ-variables are declared by an encapsulation $\{\{A^\alpha\}\}$, $\{\{B^\beta\}\}, \ldots$ and bound by the μ-abstraction $\mu\alpha.t$.

By mutual simulation with Pym's and Ritter's $\lambda\mu\nu$-calculus [29], which is sound and complete for propositional logic, we have proven the following property (see [4]).

Theorem 1. *The calculus $\lambda\mu$PRL is sound and complete for full propositional logic.*

The conversion theory for $\lambda\mu$PRL-terms is largely induced by our reference to CTT.

- Since $\lambda\mu$PRL terms are untyped, 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.
- CTT employs a lazy evaluation strategy with β-reduction. The purpose is to ensure termination without restricting the expressiveness of its type system. As $\lambda\mu$PRL is intended to be as close as possible to CTT, we will use β-reduction in a call-by-name-regime for $\lambda\mu$PRL as well.

Accordingly we define a confluent $\beta\mu$-reduction whose notion of normal form is less strong than in other $\lambda\mu$-systems. In [4] we propose two sets of permutative conversions, which might allow to us acquire the desirable proof theoretic properties of de Groote's full propositional $\lambda\mu$ [12]. $\beta\mu$-reduction for $\lambda\mu$PRL terms is defined as union of the two reduction relations defined in figure 2. For a simpler presentation we use the following abbreviations:

$$\begin{aligned} \mathit{spread}(t; x, y.u) &\equiv \texttt{let}\ \langle x, y\rangle = t\ \texttt{in}\ u \\ \mathit{decide}(t; x.u; y.v) &\equiv \texttt{case}\ t\ \texttt{of}\ \texttt{inl}(x) : u \mid \texttt{inr}(y) : v \end{aligned}$$

Proposition 1. *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. By an extended version of the usual parallel reduction method, following the approach by Baba, Hirokawa and Fujita in [2]. See [4] for details.

4 Constructive and Non-constructive Terms of $\lambda\mu$PRL

In this section we will analyze properties of normal proofs that will help us determine whether they rely on classical or constructive arguments or both. This would enable a proof system to distinguish classical from constructive proofs on a meta-level.

Our first step is to investigate in which cases $\lambda\mu$PRL-proofs are non-constructive and how this is reflected in the proof terms. Obviously, an intuitionistic fragment of $\lambda\mu$PRL could simply be obtained by removing the rules `abort1` and `abort2`. But classifying proofs that use these rules as non-constructive would exclude proofs that are constructively valid, as the application of these rules may not have been relevant for the proof. Consequently a more detailed analysis of the proof structure is needed.

If we 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 that moves formulae into the hypotheses is implication introduction, represented by a λ-abstraction in the proof term. 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" a formula in the succedent: implication elimination and disjunction introduction. This is a usual property of these rules in single-conclusioned Gentzen systems.

In the presence of the `abort`-rules, formulae that are "lost" in the intuitionistic case can be saved by moving them to the context before one of the "destructive" rules is applied and recovering them later. If the μ-variable of the recovered formula is α, the application of `abort` would result in a subterm of the form $[\alpha]t$. There are two possible cases that would render the proof classical.

1. The subtree rooted in a disjunction introduction contains a leaf that proves one disjunct or one of its subformulae using a hypothesis obtained from the other disjunct. In this case the proof term will contain a subterm
   $\mu\alpha.\ \ldots inl(\ldots \lambda x.\ \ldots [\alpha] \ldots inr(\ldots x))$ or
   $\mu\alpha.\ \ldots inr(\ldots \lambda x.\ \ldots [\alpha] \ldots inl(\ldots x))$.
   An example for this case is the standard proof of the law of the excluded middle $A \vee (A \supset \bot)$, which is represented by the term $\mu\alpha.[\alpha]inr(\lambda x.[\alpha]inl(x))$.
2. The subtree in which the premiss of an eliminated implication is to be proven does not prove this premiss but a formula from the context and depends on a hypothesis obtained from the premiss (if the context formula was proven without that hypothesis the eliminated implication would have been superfluous for the proof and been removed during cut elimination). This scenario will result in a subterm
   $\mu\alpha.\ \ldots (t\ (\ldots \lambda y.\ \ldots [\alpha] \ldots y))$
   where t is the term that represents the eliminated implication the premiss of which is proven by a switch of the goal formula. Examples are the axiom of double negation $\neg\neg A \supset A$ 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.

Now the question is how to identify such structures in proof terms. One possibility is an adaption of a method by Parigot [27]. Since in his second order $\lambda\mu$-calculus the usual formula describing natural numbers is 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 symmetric reduction rule of some $\lambda\mu$-calculi (e.g. [25]) and also to the direct extraction of computational content from classical proofs [22]. In [27] 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. Adapting the output of constructive content might allow to distinguish between proof terms with and without such content. But in this approach, the intrinsic non-determinism of classical proofs would require special care.

Another method presented by Crolard [9] provides a constructive restriction $\overline{CND}^{r}_{\vee}$ of Parigot's $\lambda\mu$ extended by disjunction and cut, which annotates the inference rules in order to define a set of interdependencies between hypotheses and conclusions. The intuition is to keep the multi-conclusion and the potential to exchange the active formula,

but to restrict the implication introduction in a way that only the hypotheses linked to the active conclusion can be used. Additionally, Crolard defines an extension of classical $\lambda\mu$ by disjunction and cut as $\lambda\mu+$. To determine if 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 $\overline{CND}^r_\vee$.

We proceed in a similar fashion by giving a constructive restriction of $\lambda\mu$PRL with annotations and defining the scope of a μ-variable for $\lambda\mu$PRL. We adopt Crolard's notation and use the following abbreviation for sets U, V, and W.

$$U[V/W] = \begin{cases} U \setminus W \cup V & if \;\; W \cap U \neq \emptyset \\ U & otherwise \end{cases}$$

Definition 1. *(Scope of a μ-variable) The set $S_[](t)$ of free λ-variables that occur* out of the scope of any *μ-variable in t and the set $S_\delta(t)$ of free λ-variables that occur* within *the scope of a free μ-variable δ in t are defined inductively on $\lambda\mu$PRL-terms.*

- $S_[](x) = x$ $\qquad S_\delta(x) = \emptyset$
- $S_[](\lambda x.u) = S_[](u) \setminus \{x\}$ $\qquad S_\delta(\lambda x.u) = S_\delta(u) \setminus x$
- $S_[](u\ v) = S_[](u) \cup S_[](v)$ $\qquad S_\delta(u\ v) = S_\delta(u) \cup S_\delta(v)$
- $S_[]([\alpha]u) = \emptyset$ $\qquad S_\delta([\alpha]u) = S_\delta(u)\ for\ any\ \delta \neq \alpha$
  $S_\alpha([\alpha]u) = S_\alpha(u) \cup S_[](u)$
- $S_[](\mu\alpha.u) = S_\alpha(u)$ $\qquad S_\delta(\mu\alpha.u) = S_\delta(u)$
- $S_[](\langle u, v\rangle) = S_[](u) \cup S_[](v)$ $\qquad S_\delta(\langle u, v\rangle) = S_\delta(u) \cup S_\delta(v)$
- $S_[](let\ \langle x, y\rangle = z\ in\ u) = S_[](u)[S_[](z)/\{x, y\}]$
  $S_\delta(let\ \langle x, y\rangle = z\ in\ u) = S_\delta(u)[S_[](z)/\{x, y\}] \cup S_\delta(z)$
- $S_[](inl(u)) = S_[](u)$ $\qquad S_\delta(inl(u)) = S_\delta(u)$
  $S_[](inr(u)) = S_[](u)$ $\qquad S_\delta(inr(u)) = S_\delta(u)$
- $S_[](case\ z\ of\ inl(x) : u \mid inr(y) : v) = S_[](u)[S_[](z)/\{x\}] \cup S_[](v)[S_[](z)/\{y\}]$
  $S_\delta(case\ z\ of\ inl(x) : u \mid inr(y) : v)$
  $= S_\delta(u)[S_[](z)/\{x\}] \cup S_\delta(v)[S_[](z)/\{y\}] \cup S_\delta(z)$

Remark 1. The above definition of the scope requires cuts to be eliminated from the term t since we cannot distinguish if a λ-abstraction resulted from a cut or from an implication introduction. This is not a serious problem as evaluating a term to check its constructivity can be combined with reduction. We could also deal with terms containing cuts if we were to use a different term to represent the application of cuts (e.g. the usual $let\ x{=}v\ in\ u$) and adapt the above definition and the reduction relation.

Now we can define μ-safety for terms of $\lambda\mu$PRL analogously to Crolard:

Definition 2. *(μ-safety of $\lambda\mu$PRL-terms) A term t of $\lambda\mu$PRL is safe in respect to μ-contexts, or μ-safe, iff for any subterm of t of the form $\lambda x.u$, $x \notin S_\delta(u)$ for any free μ-variable δ.*

It is easy to see that in both cases of the above analysis of non-constructive proofs the $\lambda\mu$PRL-terms are not μ-safe.

To verify that μ-safe terms are indeed the ones that always correspond to constructive proofs, we formulate a constructive restriction of $\lambda\mu$PRL named $\lambda\mu_r$PRL. Since this version requires much notation, we only present the basic idea and refer the interested reader to [4] for further detail.

As a starting point we use Crolard's calculus $\overline{CND}_\vee$ [9] again. In his formulation of $\overline{CND}_\vee$ Crolard annotates the formulae in the conclusion by their interdependencies with λ-variables. These sets of interdependencies are exactly the sets from definition 1, 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 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 passive formula – otherwise the variable would occur in the scope of the μ-variable associated with the respective formula.

As our calculus works in top-down direction, we need some more notation than in a bottom-up calculus. The inductive definition of the scope, and equivalently the interdependencies, means that the scopes of μ-variables can – like the proof term – be computed only when the proof is completed. Thus, when an implication introduction is applied, there is not enough information on the scopes to restrict the rule as Crolard does. However, we can implement this restriction by additional annotation. We simply add a set 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 which λ-variables in a term may not occur freely in its subterms.

The sequents of $\lambda\mu_r$PRL are $\lambda\mu$PRL-sequents with the following extensions: 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 corresponding to the rules applied reflects and is consistent with definition 1. This draws the connection between the calculus and the concept of μ-safety.

The rule `lambdaI` (as well as the rules `functionE`, `andI`, `andE` and `orE`) requires us to express operations on the annotation of each μ-variable. Let

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

An excerpt from the rules of $\lambda\mu_r$PRL can be found in figure 3.

We can establish that μ-safe $\lambda\mu$PRL-terms indeed correspond to proofs in $\lambda\mu_r$PRL and that $\lambda\mu_r$PRL is sound and complete for intuitionistic propositional logic.

Proposition 2. *A $\lambda\mu$PRL-term corresponds to a proof in $\lambda\mu_r$PRL iff it is μ-safe.*

$\texttt{G, } x{:}T\texttt{, H} \vdash \{x\}|\mathcal{F}{:}T \texttt{ ext } x$
BY hypothesis $\$i$
$if\ x \notin \mathcal{F}$
(no subgoals)

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

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

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

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

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

Fig. 3. Excerpt from $\lambda\mu_r$PRL's inference rules

Proof.

$\Rightarrow$ We have to show for a $\lambda\mu_r$PRL-term t and any subterm of the form $\lambda x.u$ that $x \notin S_\delta$ for any μ-variable δ occurring freely in u.
We know that in $\lambda\mu_r$PRL the application of `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` 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 $x \notin S_\delta$ holds for any subterm of the form $\lambda x.u$ an any μ-variable δ occurring freely in u,. Therefore x is not used by the `hypothesis`-rule in any of the subproofs corresponding to subterms of the form $[\delta]v$ of $\lambda x.u$ 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. This can be done by simulating its rules in Nuprl or a calculus equivalent to its propositional subsystem.

Theorem 2. *$\lambda\mu_r$PRL is sound and complete for intuitionistic propositional logic.*

Proof. This can be shown by mutual simulation of $\lambda\mu_r$PRL and any sound and complete calculus for intuitionistic logic. The idea behind the simulation is as follows:

$\Rightarrow$ 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 that 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 $\vdash (A \supset D) \vee (B \supset C)$. Then we can prove all rules of $\lambda\mu_r$PRL as lemmas.

$\Leftarrow$ The propositional inference rules of an equally expressive constructive logic (e.g. the propositional fragment of Gentzen's $\mathcal{LJ}$ or CTT) are obviously theorems of $\lambda\mu_r$PRL as there is no need for context formulae and thus the set $\mathcal{F}$ trivially remains empty throughout the proofs of the rules.

Now we can use the concept of μ-safety for our original purpose and evaluate a term t of $\lambda\mu$PRL in two steps: Computing the scopes of the free μ-variables present in t and 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.

We can also compute the constructive λ-term from a μ-safe $\lambda\mu$PRL-term by evaluating the μ-operator like Felleisen's $\mathcal{C}$-operator (see section 5) but that requires a deterministic evaluation strategy. Otherwise such an evaluation would not be confluent.

For $\lambda\mu$PRL-terms in general, 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).

In the case of μ-safe terms any strategy will yield a constructive term but the resulting term may depend on the strategy. In contrast to that, for non-μ-safe terms the evaluation might produce a non-closed term (which reflects that the crucial hypotheses are not available in an intuitionistic proof).

There is another, somewhat pathological subset of $\lambda\mu$PRL-terms which are not μ-safe but may evaluate to a constructive, closed λ-term depending on the evaluation order. Thus, in the above example, $C[u']$ might be constructive whereas $C[v']$ is not.

The question is when this may be the case. 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 define a notion of *partial μ-safety*.

Definition 3. *(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 when using the appropriate evaluation order.

5 Relation to Murthy's $PROG_K$

Like $\lambda\mu$PRL, Murthy's calculus $PROG_K$ [22] has been developed in the context of Computational Type Theory and provides an extension to classical logic. As a consequence $\lambda\mu$PRL and $PROG_K$ share the following properties:

- both use logical rules of CTT
- 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 $\bot$.

Therefore it is interesting to look into the exact relation between $\lambda\mu$PRL and $PROG_K$.

Murthy's work is primarily concerned with the direct extraction of constructive content from proofs of Σ-sentences in Peano Arithmetic. Thus, if we adapt $\lambda\mu$PRL in an appropriate way, it is also possible to reproduce Murthy's results for this purpose. Such an adaption has to overcome the following differences:

- $PROG_K$ 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 theorems are restricted to the decidable fragment Σ_1^0 (respectively Π_2^0).
- $PROG_K$ is obtained from its intuitionistic fragment by adding the double negation elimination rule. It types Felleisen's $\mathcal{C}$-operator [13]. When reducing $\mathcal{C}$, the abort-operator $\mathcal{A}$ (also due to Felleisen) is used.
- 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 functional programming language with access to the control flow. Because of the known confluence issues, Murthy defines a deterministic evaluation strategy.

The close relation between $\lambda\mu$PRL and $PROG_K$ can be shown similarly to how de Groote [10] relates a first order version of Parigot's $\lambda\mu$ to a subtheory of Felleisen's syntactic theory of sequential control [13]. We will focus on the basic idea of translating between an adapted version of $\lambda\mu$PRL and $PROG_K$ and refer to [4] for details.

Since $\lambda\mu$PRL and $PROG_K$ do not cover the same range of formulae, we formulate a variant $\lambda\mu_c$PRL of $\lambda\mu$PRL that extends the logic by the arithmetical and first-order expressions of $PROG_K$ as well as the accompanying rules. For further comparability with $PROG_K$ and to maintain soundness, $\lambda\mu_c$PRL is restricted to decidable formulae and to the type $\mathbb{N}$.

The deterministic evaluation in $PROG_K$ is based on the following two reduction rules, using the notion of evaluation contexts $E[-]$ with a hole:

Definition 4. *(c-reduction)*

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

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. For this purpose, the $\mathcal{A}$-operator needs to be included the term-language of $\lambda\mu_c$PRL.

Definition 5. *(μ_c-reduction)*

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

To make the evaluation comparable, the evaluation strategy is assumed to be the same as in $PROG_K$.

Since the only crucial difference between $PROG_K$ and $\lambda\mu_c$PRL arises from the "classical" constructs μ and $\mathcal{C}$, it is sufficient to look at their relation to get a basic idea of the relation between the two calculi. We start with defining a $\mathcal{C}$-transform $[\![t]\!]^{\mathcal{C}}$ of a $\lambda\mu_c$PRL-term t. This is done inductively and the interesting cases are:

- $[\![\mu\alpha.t]\!]^{\mathcal{C}} = (\mathcal{C}\ \lambda\alpha.[\![t]\!]^{\mathcal{C}})$;
- $[\![[\alpha]t]\!]^{\mathcal{C}} = (\alpha\ [\![t]\!]^{\mathcal{C}})$;
- $[\![(\mathcal{A}\ t)]\!]^{\mathcal{C}} = (\mathcal{A}\ [\![t]\!]^{\mathcal{C}})$;

Based on this definition we can show the following proposition (See [4] for a proof):

Proposition 3.
If t is a $\lambda\mu_c$PRL-term and A is a simple type such that $\vdash_\mu A$ `ext` t then $\vdash_{\mathcal{C}} A$ `ext` $[\![t]\!]^{\mathcal{C}}$.

The inverse translation is based on the proof of the double negation elimination rule $\neg\neg A \supset A$ in $\lambda\mu_c$PRL. Applying the extract term of the proof to the translated subterm t yields the μ-transform of a $PROG_K$-term $(\mathcal{C}\ t)$. So we can proceed with defining the μ-transform $[\![t]\!]^{\mu}$ of a $PROG_K$- term t. Again we omit the less illuminating cases:

Definition 6. *(μ-transform)*
$[\![(\mathcal{C}\ t)]\!]^{\mu} = \mu\alpha.([\![t]\!]^{\mu}\ \lambda y.[\alpha]y)$ $\qquad$ $[\![(\mathcal{A}\ t)]\!]^{\mu} = (\mathcal{A}\ [\![t]\!]^{\mu})$

Based on this definition we can show by an induction on the derivation of $\vdash_{\mathcal{C}} A$ `ext` t:

Proposition 4.
If t is a $PROG_K$-term and A is a simple type such that $\vdash_{\mathcal{C}} A$ `ext` t then $\vdash_\mu A$ `ext` $[\![t]\!]^{\mu}$.

Remark 2. A translation of the operator $\mathcal{A}$ is not necessary for the translation of derivations in the two calculi, as it will not occur in a derivation. However, it is necessary for evaluation purposes.

Based on the above translations some results involving the evaluation of terms and thus of computational content can be established (see [4] for proofs of propositions 5–7). We denote the evaluation of $\lambda\mu_c$PRL that combines Murthy's reduction strategy with β- and μ_c-reduction by $ev_{\overrightarrow{\mu}}$ and the evaluation of $PROG_K$ by $ev_{\overrightarrow{c}}$.

Proposition 5. *Let t_1 and t_2 be $\lambda\mu_c$PRL-terms. If $t_1 ev_{\overrightarrow{\mu}}\ t_2$ then $[\![t_1]\!]^{\mathcal{C}} ev_{\overrightarrow{c}}\ [\![t_2]\!]^{\mathcal{C}}$.*

Proposition 6. *Let t_1 and t_2 be $PROG_K$-terms. If $t_1 ev_{\overrightarrow{c}}\ t_2$ then $[\![t_1]\!]^{\mu} ev_{\overrightarrow{\mu}}\ [\![t_2]\!]^{\mu}$.*

The reduction rules combined with the evaluation strategy induce an equality relation for each of the two calculi. Using this equality relation on terms, we can also show that the two translations are inverses of each other.

Proposition 7. *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. $[\![[\![t]\!]^{\mu}]\!]^{\mathcal{C}} =_c t$

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

Theorem 3. *If t_1, t_2 are closed $\lambda\mu_c$PRL-terms and s_1, s_2 are $PROG_K$-terms 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 5, 6 and 7.

6 Conclusions and Future Work

We have shown that it is possible to distinguish between "constructive" and "non-constructive" $\lambda\mu$PRL-terms by a detailed analysis of the term structure. The notions of μ-safe and partially μ-safe terms can be used to implement a procedure that identifies proof terms with constructive content even if the proof contained `abort`-rules.

As a consequence of the isomorphism between $\lambda\mu_c$PRL and $PROG_K$ we can also directly extract computational content from proofs in $\lambda\mu_c$PRL. Using the same evaluation for $\lambda\mu$PRL, however, might result in non-closed terms since there is no restriction on the goal formulae. This will be the case if the proof relies on a hypothesis not available in an constructive proof. In the case of μ-safe terms, an intuitionistic proof term can be recovered using any evaluation strategy since no subgoal may depend on a "classical" hypothesis. Partially μ-safe terms will evaluate to a closed λ-term for at least one evaluation sequence.

Still, we cannot conclude from a proof term which does not evaluate as constructive that there is no constructive proof of the proposition. The result simply states that the proof in question relies on a classical argument. Considering the relation between classical logic and control operators in functional programming languages (and thus imperative programming constructs), it would be interesting to further examine the computational content of such "classical" terms.

There are various potential directions for future work. 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 followed the design decisions of CTT, a basic implementation could be done quite straightforwardly. All rules except the rules `mu` and `abort` already exist in Nuprl and the new constructs could be easily embedded as (abstraction-, rule-, and ML-) objects in Nuprl's library. As the proof term of Nuprl proofs is computed after the completion of a proof, this procedure might be well suited to also include further evaluation of the proof term to decide whether the proof is constructive in the intuitionistic sense or not.

It would also be interesting to further examine the idea of partial μ-safety, e.g. establishing whether the set of partial μ-safe terms contains every proof term with constructive content. The concept could be extended to $\lambda\mu_c$PRL, possibly allowing to drop the restriction on its goal formulae by subsequently identifying non-constructive terms.

Extending $\lambda\mu$PRL to first order logic and arithmetic cannot be done without restricting it to decidable formulae (instead of permitting the use of the axiom of choice). In CTT, the existential quantifier is defined as the dependent product 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, using 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 and prove something else. In fact, it has been shown in [21] that an arithmetic type theory that contains the dependent product type and validates the principle of the excluded middle is unsound. The essential reason is that the undecidable halting problem could be proven decidable in such a theory.

Nevertheless, if we could identify non-constructive $\lambda\mu_c$PRL-proofs, this problem could be circumvented. We could then employ the meta-system in a fashion similar to

Howe's method for importing proofs from HOL into Nuprl [18]. By using Nuprl's meta-system to mark a proof's root node whenever it is not computational, Howe made sure that proofs containing instances of HOL's non-constructive "select"-operator cannot be used when computation is required. Investigating the relation between HOL's select-operator and $\lambda\mu$PRL's μ-operator and studying how Howe's operational semantics for CTT can be reconciled with $\lambda\mu$PRL could provide a basis for adapting his method to $\lambda\mu_c$PRL-proofs.

Besides, it is well known that the μ-operator can be seen as "generic jump operator" [25]. Therefore the terms of $\lambda\mu$PRL correspond to functional programming languages with control operators. In [25] Ong explicitly describes such a correspondence between terms of μPCF and different kinds of control constructs: the `Y`-combinator, Scheme's `call/cc` and ML-style exception handling via `throw` and `catch`. Other authors have also presented calculi for λ-calculi with control, like de Groote's calculus for exception handling [11], which is closely related to $\lambda\mu$.

Along these results, an extension of Nuprl's purely functional programming language might be of interest instead of merely employing the μ-operator to distinguish between intuitionistic and classical proofs. It could be considered how to define different control constructs as conservative extensions based on the μ-operator and whether a semantics of evidence for a hybrid type theory could be developed on this foundation.

References

1. Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *Journal of Applied Logic* 4(4):428–469, 2006.
2. Kensuke Baba, Sachio Hirokawa, and Ken-etsu Fujita. Parallel reduction in type free $\lambda\mu$-Calculus. *Electronic Notes on Theoretical Computer Science* 42, 2001.
3. Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Program extraction from classical proofs. *Annals of Pure and Applied Logic*, pages 77–97, 1995.
4. Nuria Brede. $\lambda\mu$PRL – A Proof Refinement Calculus for Classical Reasoning in Computational Type Theory Diploma thesis, Institut für Informatik, Universität Potsdam, 2009. available at: `http://www.cs.uni-potsdam.de/~brede`
5. Robert L. Constable. Computational type theory. Scholarpedia, 4(2):7618, 2009.
6. Robert L. Constable et. al. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, 1986.
7. S. A. Cook. The complexity of theorem proving procedures. *STOC-71*, pp. 151–158, 1971.
8. Thierry Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60(1):325–337, 1995.
9. Tristan Crolard. A constructive restriction of the $\lambda\mu$-calculus. Technical Report 02, UFR d'Informatique, Université Paris 7, 2002.
10. Philippe de Groote. On the relation between the lambda-mu-calculus and the syntactic theory of sequential control. *LPAR-94*, pages 31–43, 1994.
11. Philippe de Groote. A simple calculus of exception handling. *TLCA*, pages 201–215, 1995.
12. Philippe de Groote. Strong normalization of classical natural deduction with disjunction. *TLCA*, pages 182–196, 2001.
13. Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. A syntactic theory of sequential control. *Theoretical Compututer Science* 52:205–237, 1987.

14. 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.
15. Michael J. C. Gordon and Thomas F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
16. Timothy Griffin. A formulae-as-types notion of control. *POPL-90*, pages 47–58, 1990.
17. 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, 1980.
18. Douglas J. Howe. Semantic Foundations for Embedding HOL in Nuprl. *5th International Conference on Algebraic Methodology and Software Technology*, pages 85–101, 1996.
19. Christoph Kreitz. *The Nuprl Proof Development System, Version 5, Reference Manual and User's Guide*. Cornell University, 2002.
20. Per Martin-Löf. *Intuitionistic Type Theory*. *Studies in Proof Theory 1*, Bibliopolis, 1984.
21. Chetan R. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Department of Computer Science, Cornell University, 1990.
22. Chetan R. Murthy. An evaluation semantics for classical proofs. *LICS '91*, pp. 96–107, 1991.
23. Koji Nakazawa and Makoto Tatsuta. Strong normalization of classical natural deduction with disjunctions. *Annals of Pure and Applied Logic*, 153:21–37, 2008.
24. C.-H. L. Ong. A semantic view of classical proofs. *LICS '96*. IEEE Press, 1996.
25. C.-H. L. Ong and C. A. Stewart. A curry-howard foundation for functional computation with control. *ACM SIGPLAN-SIGACT Symposium on Principle of Programming Languages*, pages 215–227. ACM Press, 1997.
26. Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. *LPAR-92*, pages 190–201, 1992.
27. Michel Parigot. Classical proofs as programs. *Computational logic and proof theory*, LNCS 713, pages 263–276. Springer-Verlag, 1993.
28. Walter Py. *Confluence en $\lambda\mu$-calcul*. PhD thesis, UFR Sciences Fondamentales et Appliquées, Université de Savoie, 1998.
29. D. J. Pym and E. Ritter. *Reductive Logic and Proof-search*. Oxford University Press, 2004.
30. D. J. Pym and E. Ritter. On the semantics of classical disjunction. *Journal of Pure and Applied Algebra*, 159:315–338, 2001.
31. Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Compututer Science* 9:67–72, 1979.