Cut Formulae and Logic Programming

Luis Pinto *

luis@dcs.st-and.ac.uk
Computer Science Division
University of St Andrews
Scotland

Abstract. In this paper we present a mechanism to define names for proof-witnesses of formulae and thus to use Gentzen's cut-rule in logic programming. We consider a program to be a set of logical formulae together with a list of such definitions. Occurrences of the defined names guide the proof-search by indicating when an instance of the cut-rule should be attempted. By using the cut-rule there are proofs that can be made dramatically shorter. We explain how this idea of using the cut-rule can be applied to the logic of hereditary Harrop formulae.

1 Introduction

The computation mechanisms both for logic and for functional programming are searches for cut-free proofs. First, in pure logic programming the achievement of a goal G w.r.t. a program P can be seen[1] as the search for a proof in Gentzen's intuitionistic sequent calculus LJ [Gen69], of the sequent $P \Rightarrow G$, that by Gentzen's cut-elimination theorem can be cut-free [Bee89], [Mil90]; a λ-term found as a witness to a proof contains among other things the answer substitution. Second, the conventional view of functional programming, as in [Tho91], is that one constructs a sequence of definitions and an expression to be evaluated; the evaluation of the expression is done by replacing the definienda by their definientia and subsequent normalisation. By the Curry-Howard correspondence between types and propositions, the evaluation of expressions in functional programming corresponds to the normalisation of proofs in Gentzen's natural deduction system NJ. So, both processes yield cut-free proofs, using "cut-free" first in the sequent calculus sense and second in the natural deduction sense.

* Supported by JNICT (Portugal) grant BD/1423/91-IA and by ESPRIT grant BRA 7232 "GENTZEN".

[1] This view is contrary to the one expressed in [GLT89], where a program is a set of sequents and the achievement of a goal w.r.t. a program is the search for a proof using instances of the sequents in the program as proper axioms and the cut-rule as the only inference rule. The latter approach does not generalise when we allow for non-Horn formulae in the program.

From a type-theoretic point of view [ML84], [Tho91], in logic programming we give a specification (goal formula) and we want to find an object meeting that specification, and in functional programming we give a specification (type) and an object (the expression to be evaluated) meeting the specification and we want to transform the object into another object in normal form meeting the specification.

If we were to allow the cut-rule in proof-search some formulae would have much shorter proofs. For example, in [Boo84] it is shown that there are formulae whose shortest cut-free proofs are exponentially longer than their shortest proofs using instances of the cut-rule. The problem of using the cut-rule is to decide when and how the cut-rule should be applied; in other words, what are the adequate lemmas to use in proving a theorem. The lemmas are usually established based on experience. In programming, we do not expect to establish lemmas during the proof-search. Instead we expect the programmer to know what lemmas may be useful and define names for the proofs of these lemmas. Then, during the search for a proof of a formula we can use these formulae previously established without having to prove them several times. As a result of proof-search in this framework we can obtain proofs with instances of the cut-rule; if required cut-elimination can be applied. Although we allow for instances of the cut-rule during the search for a proof, proof-search can still be efficient since the application of the cut-rule only needs to be attempted in particular circumstances, to be described in full below. Briefly, the cut-rule is attempted if there is an occurrence of a defined name in the goal in which case the type of the defined name is used as cut formula.

In our system we have two layers of typed objects. On one layer the objects are λ-terms with constants and their types are the simple types of Church's theory of types. On the other layer the objects are proof-witnesses and their types are logical formulae. Terms may occur both in proof-witnesses and logical formulae but occurrences of proof-witnesses in the logical formulae are not allowed.

This paper is organized as follows. We start by introducing the underlying language of typed λ-calculus with constants and by presenting the calculus LJ^a for intuitionistic logic with proof-witnesses annotating the formulae. Next, we describe the idea of using definitions to control the proof-search and we apply this idea to the logic of hereditary Harrop formulae. An example of how this technique may be used to find shorter proofs follows. Finally we mention other related works and present some concluding remarks.

2 Logical Preliminaries

We will introduce the underlying language of typed λ-calculus with constants based on [Mil90]. Let us consider a type system with a set $\mathcal{S}$ of *primitive types*. We assume the symbol o, the type of *propositions*, to be a member of $\mathcal{S}$. The set of *types* is the closure of $\mathcal{S}$ under the formation of function types. The constructor of function types is denoted by $\rightarrow$ and it associates to the right. The symbols $\tau, \tau_0, ...$ are used to denote arbitrary types. Any type τ can be written as $\tau_1 \rightarrow ... \rightarrow \tau_n \rightarrow \tau_0$, where τ_0 is a primitive type. In the particular case where $n = 0$ the type τ is just τ_0.

We assume there is a set $\mathcal{C}$ of typed *constants* of and a set $\mathcal{X}$ of denumerably many *variables* of each type. We also assume there is a set $\mathcal{A}$, whose members we call *parameters*, with denumerably many parameters of each type. We use $a, a_1, a_2, ...$ as parameters. A *signature* Σ is a set, whose elements are either constants or parameters. We often display the members of a signature as pairs $s : \tau$, where s is either a constant or a parameter and τ is its type. Terms are built up from constants, parameters and variables by application and abstraction over variables as usual, subject to the type rules. For example, if x is a variable of type τ, a is a parameter of type τ and c is a constant of type $\tau \rightarrow \tau \rightarrow \tau$ then $\lambda x.cax$ is a term of type $\tau \rightarrow \tau$. An *atomic formula* A is a term of the form $pt_1...t_n$, where p is a constant of type $\tau_1 \rightarrow ... \rightarrow \tau_n \rightarrow o$; A is a *first-order atomic formula* if $\tau_1, ..., \tau_n$ are primitive types different from o. *(Logical) formulae* are built up from atomic formulae by using the *logical constants* $\wedge : o \rightarrow o \rightarrow o$, $\vee : o \rightarrow o \rightarrow o$, $\supset$: $o \rightarrow o \rightarrow o$, and for every type τ, $\exists_\tau : (\tau \rightarrow o) \rightarrow o$ and $\forall_\tau : (\tau \rightarrow o) \rightarrow o$. We use the infix notation $t_1 \wedge t_2$, $t_1 \vee t_2$, $t_1 \supset t_2$ to display $\wedge t_1 t_2$,$\vee t_1 t_2$,$\supset t_1 t_2$ respectively, and we display formulae of the form $\exists_\tau(\lambda x.t)$,$\forall_\tau(\lambda x.t)$ as $\exists_{x:\tau} t$,$\forall_{x:\tau} t$ respectively.

As usual, an occurrence of a symbol s within a term can be classified as either *bound* if s occurs in the scope of λs or *free* otherwise. A term is *closed* if it contains no free variable occurrences. We use $[t_1/x]t$ to indicate the term obtained from t by replacing the free occurrences of the variable x by the term t_1 changing bound variable names to avoid variable capture. We define *$(\alpha)\beta\eta$-convertibility* as usual and we identify terms that are α-convertible. Normally we abbreviate $(\alpha)\beta\eta$-convertibility by *λ-convertibility*. A term is in *normal form* if it contains no occurrences of β- or η-redexes. For every term t, t has a unique normal form that we write $\lambda norm(t)$.

Let Σ be a signature. A *Σ-term* is a term all of whose symbols occurring freely are members of Σ; in other words, a Σ-term is a closed term all of whose constants and parameters are in Σ. A *Σ-formula* is a formula all of whose nonlogical constants occurring freely are members of Σ.

Let $\mathcal{U}$ be a denumerable set, whose members $W, W_1, W_2, \ldots$ we call *dummies*. Let $\mathcal{R}$ be a denumerable set, whose members $r, r_1, r_2, \ldots$ we call *abstract realisers*.

The set w of *proof-witnesses* is inductively defined as follows:

$$\begin{aligned} w ::= {} & \pi_1 w \mid \pi_2 w \mid \langle w, w\rangle \mid inl\, w \mid inr\, w \mid r \mid when\, www \mid ww \mid \lambda r.w \\ & \mid \langle t, w\rangle \mid wt \mid \lambda a.w \mid let\ r = w\ in\ w \mid let\ a = \pi_1 r\ in\ w \mid W, \end{aligned}$$

where r ranges over $\mathcal{R}$, a ranges over $\mathcal{A}$, W ranges over $\mathcal{U}$ and t ranges over closed terms. We use $w, w_1, w_2, \ldots$ to write proof-witnesses.

An occurrence of a parameter a in a proof-witness is *free* if it is not in the scope of λa or $let\ a$; it is *bound* otherwise. Let Σ be a signature. A proof-witness w is a *Σ-proof-witness* if all its constants are in Σ and if all the parameters occurring freely in w are also in Σ.

A *sequent* is a quadruple $\Sigma; \Gamma \Rightarrow w : F$, where

1. Σ is a signature;
2. Γ is a finite set of pairs $r_1 : F_1, \ldots, r_n : F_n$, where $r_1, \ldots, r_n$ are distinct abstract realisers and $F_1, \ldots, F_n$ are Σ-formulae;
3. w is a Σ-proof-witness;
4. F is a Σ-formula.

The set Γ is called the *antecedent* of the sequent and the formula F is called the *succedent* of the sequent. Informally we read a sequent as: the term w witnesses the provability of the goal F w.r.t. the set of assumptions Γ over signature Σ.

Figure 1 presents the sequent calculus LJ^a for intuitionistic logic over typed λ-terms with proof-witnesses annotating the formulae.

Theorem 1. *Let $\Sigma; \Gamma \Rightarrow w : G$ be a provable sequent in LJ^a. Then there is a proof-witness w_1 s.t. the sequent $\Sigma; \Gamma \Rightarrow w_1 : G$ has a proof with no instances of the cut-rule.*

Proof. By induction on the structure of the proof of the sequent $\Sigma; \Gamma \Rightarrow w : G$ and the size of the cut formula as usual. □

A *uniform proof*, as defined in [MNPS91], is a proof in which each occurrence of a sequent whose succedent is nonatomic is the conclusion of the rule that introduces its top-level connective.

3 Proof-Search Using the Cut-Rule

By Theorem 1, if we avoid using the cut-rule of LJ^a we can still prove all the sequents we could prove before. However, there are then proofs

$$\frac{}{\Sigma;\Gamma, r:A \Rightarrow r:A}\ axiom$$

$$\frac{\Sigma; r_1:A, r_2:B, \Gamma \Rightarrow w:C}{\Sigma; r:A\wedge B, \Gamma \Rightarrow let\ r_1 = \pi_1 r\ in\ let\ r_2 = \pi_2 r\ in\ w:C}\ \wedge\Rightarrow$$

$$\frac{\Sigma;\Gamma \Rightarrow w_1:A \quad \Sigma;\Gamma \Rightarrow w_2:B}{\Sigma;\Gamma \Rightarrow \langle w_1, w_2\rangle : A\wedge B}\ \Rightarrow\wedge \qquad \frac{\Sigma;\Gamma \Rightarrow w:A}{\Sigma;\Gamma \Rightarrow inl\, w : A\vee B}\ \Rightarrow\vee_1$$

$$\frac{\Sigma;\Gamma \Rightarrow w:B}{\Sigma;\Gamma \Rightarrow inr\, w : A\vee B}\ \Rightarrow\vee_2 \qquad \frac{\Sigma;\Gamma, r_1:A \Rightarrow w_1:C \quad \Sigma;\Gamma, r_2:B \Rightarrow w_2:C}{\Sigma;\Gamma, r:A\vee B \Rightarrow when\, r(\lambda r_1.w_1)(\lambda r_2.w_2):C}\ \vee\Rightarrow$$

$$\frac{\Sigma;\Gamma, r:A\supset B \Rightarrow w_1:A \quad \Sigma;\Gamma, r:A\supset B, r_1:B \Rightarrow w_2:C}{\Sigma;\Gamma, r:A\supset B \Rightarrow let\ r_1 = rw_1\ in\ w_2:C}\ \supset\Rightarrow$$

$$\frac{\Sigma; r:A,\Gamma \Rightarrow w:B}{\Sigma;\Gamma \Rightarrow \lambda r.w : A\supset B}\ \Rightarrow\supset \qquad \frac{\Sigma;\Gamma \Rightarrow w:[t/x]A}{\Sigma;\Gamma \Rightarrow \langle t, w\rangle : \exists_{x:\tau}A}\ \Rightarrow\exists$$

$$\frac{\Sigma\cup\{a:\tau\}; r_1:[a/x]A, \Gamma \Rightarrow w:B}{\Sigma; r:\exists_{x:\tau}A, \Gamma \Rightarrow let\ a = \pi_1 r\ in\ let\ r_1 = \pi_2 r\ in\ w:B}\ \exists\Rightarrow$$

$$\frac{\Sigma; r_1:[t/x]A, r:\forall_{x:\tau}A, \Gamma \Rightarrow w:B}{\Sigma; r:\forall_{x:\tau}A, \Gamma \Rightarrow let\ r_1 = rt\ in\ w:B}\ \forall\Rightarrow \qquad \frac{\Sigma\cup\{a:\tau\};\Gamma \Rightarrow w:[a/x]A}{\Sigma;\Gamma \Rightarrow \lambda a.w : \forall_{x:\tau}A}\ \Rightarrow\forall$$

$$\frac{\Sigma;\Gamma \Rightarrow w:B \quad \Sigma; r:B,\Gamma \Rightarrow w_1:A}{\Sigma;\Gamma \Rightarrow let\ r = w\ in\ w_1:A}\ cut \qquad \frac{\Sigma; r_1:A_1', ..., r_n:A_n' \Rightarrow w:A'}{\Sigma; r_1:A_1, ..., r_n:A_n \Rightarrow w:A}\ \lambda conv$$

Provisos:

1. in the $axiom$, A is an atomic formula;
2. in $\wedge\Rightarrow$ and $\vee\Rightarrow$, r_1 and r_2 are new abstract realisers;
3. in $\supset\Rightarrow$, r_1 is a new abstract realiser;
4. in $\Rightarrow\exists$, t is a Σ-term of type τ;
5. in $\Rightarrow\forall$, a is a new parameter;
6. in $\exists\Rightarrow$, a is a new parameter and r_1 is a new abstract realiser;
7. in $\forall\Rightarrow$, t is a Σ-term of type τ and r_1 is a new abstract realiser;
8. in cut, r does not occur in Γ,
9. in $\lambda conv$, $A_1', ..., A_n', A'$ are λ-convertible to $A_1, ..., A_n, A$ respectively.

Fig. 1. The calculus LJ^a.

that will be exponentially longer than if they were built up by using the cut-rule. The problem of using the cut-rule is that it does not preserve the subformula property since the cut formula, the formula B in the cut-rule of LJ^a, might not occur in the conclusion sequent and so, proof-search is difficult.

Usually in logic programming a program is a set or a list of logical formulae. In our approach we consider a program to be a set of logical formulae together with a list of definitions. The names being defined in the list of definitions will guide the search for a proof since they will be responsible for triggering instances of the cut-rule. The cut-rule is applied only in case there is an occurrence of a defined name in the goal formula. In this case the cut-rule is applied and the cut formula is the type of the defined name. Below, we explain how this idea of using the cut-rule can be applied to the logic of first-order hereditary Harrop formulae, for which uniform proofs are complete as shown in [MNPS91], and thus efficient proof-search strategies can be devised.

The[2] set H of first-order *hereditary Harrop formulae*[3] is inductively defined by:

$$\begin{aligned}H &::= A \mid H \wedge H \mid G \supset A \mid \forall_{x:\tau} H\\ G &::= A \mid G \wedge G \mid G \vee G \mid \exists_{x:\tau} G \mid H \supset G \mid \forall_{x:\tau} G,\end{aligned}$$

where the meta-variable A ranges over the set of first-order atomic formulae and τ ranges over the set of primitive types different from o. We define the set I of *I-formulae* as:

$$I ::= A \mid I \wedge I \mid I \supset A \mid \forall_{x:\tau} I,$$

where the meta-variable A ranges over the set of first-order atomic formulae and τ ranges over the set of primitive types different from o. The set of I-formulae is the intersection of the sets of H- and G-formulae. A formula is called a *C(ut)-formula* if it has the form

$$\exists_{f_1:\tau_1} \ldots \exists_{f_n:\tau_n} I,$$

where $\tau_1, \ldots, \tau_n$ are arbitrary types and I is an I-formula.

Let Σ be a signature. The set of *ΣH-formulae* is the set of all Σ-formulae that are also H-formulae. Likewise we define the sets of *ΣG-formulae* and *ΣC-formulae*.

---

[2] The letter H is used to denote hereditary Harrop formulae rather than D because we use below the letter D to denote sets of defined names.

[3] First-order hereditary Harrop formulae, as defined in [MNPS91], are formulae in λ-normal form; here we also consider formulae in non-λ-normal form.

Let $\mathcal{D}$ be a set, whose members $d, d_1, d_2, ...$ we call *defined names*. We extend the definition of Σ-proof-witnesses by allowing proof-witnesses to be built up also from defined names, i.e.

$$\begin{aligned} w ::= &\ \pi_1 w \mid \pi_2 w \mid \langle w, w\rangle \mid inl\, w \mid inr\, w \mid s \mid when\, www \mid ww \mid \lambda r.w \\ &\mid \langle t, w\rangle \mid wt \mid \lambda a.w \mid let\ s = w\ in\ w \mid let\ a = \pi_1 s\ in\ w \mid W, \end{aligned}$$

where s ranges over $\mathcal{R} \cup \mathcal{D}$.

A proof-witness of a C-formula has the form $\langle t_1, \langle t_2, ...\langle t_n, w\rangle ...\rangle\rangle$. If we define a name d for such proof-witness the expressions $\pi_1 d$, $\pi_1(\pi_2 d), ..., \pi_1(\pi_2^{n-1} d))$ denote the terms $t_1, t_2, ..., t_n$ respectively, where π_2^{n-1} represents $n-1$ applications of π_2. We define a set $\mathcal{E}$ of typed expressions of the form:

$$(\pi_1 d)^{\tau_1}, (\pi_1(\pi_2 d))^{\tau_2}, (\pi_1(\pi_2(\pi_2 d)))^{\tau_3}, ...,$$

where d ranges over $\mathcal{D}$ and $\tau_1, \tau_2, \tau_3, ...,$ range over types. We redefine the set of Σ-terms to be also allowed to be built up from expressions e^τ in $\mathcal{E}$, which are considered to be Σ-terms of type τ. Let $\mathcal{D}'$ be a subset of $\mathcal{D}$. The set of $\Sigma\mathcal{D}'$-terms is the set of Σ-terms, all of whose defined names are in $\mathcal{D}'$. For example, if $d \in \mathcal{D}'$ and $(\pi_1 d)^{\tau\to\tau} \in \mathcal{E}$ and $c : \tau \in \Sigma$ then $(\pi_1 d)^{\tau\to\tau} c$ is a $\Sigma\mathcal{D}'$-term of type τ. Likewise we define $\Sigma\mathcal{D}'$-*formulae*, $\Sigma\mathcal{D}'H$-*formulae*, $\Sigma\mathcal{D}'G$-*formulae* and $\Sigma\mathcal{D}'C$-*formulae*.

Let Σ be a signature and let $\mathcal{D}'$ be a set of defined names. We define two new sets w^+ and w^- of proof-witnesses; roughly speaking, w^--proof-witnesses will be used to annotate formulae in the program and the w^+-proof-witnesses will be used to annotate goal-formulae. The set w^+ of $\Sigma\mathcal{D}'$-w^+-*proof-witnesses* and the set w^- of $\Sigma\mathcal{D}'$-w^--*proof-witnesses* are inductively defined as follows:

$$\begin{aligned} w^+ &::= w^- \mid \langle w^+, w^+\rangle \mid inl\, w^+ \mid inr\, w^+ \mid \lambda r.w^+ \mid \langle t, w^+\rangle \\ &\qquad\qquad \mid \lambda a.w^+ \mid let\ d = w^+\ in\ w^+ \mid W, \\ w^- &::= \pi_1 w^- \mid \pi_2 w^- \mid r \mid w^- w^+ \mid w^- t \mid d, \end{aligned}$$

where r ranges over $\mathcal{R}$, a ranges over $\mathcal{A}$, W ranges over dummies, d ranges over $\mathcal{D}'$ and t ranges over $\Sigma\mathcal{D}'$-terms. We use $w, w_1, w_2, ...$ to write w^+-proof-witnesses and $w_1^-, w_2^-, ...$ to write w^--proof-witnesses.

Let Σ be a signature, d a defined name, $\mathcal{D}'$ a set of defined names not containing d, w a $\Sigma\mathcal{D}'$-w^+-proof-witness and C a $\Sigma\mathcal{D}'C$-formula. Then $d =_{def} w : C$ is called a *definition*, with *definiendum* d, *definiens* w and *type* C.

A $\Sigma\mathcal{D}'$-term t is *well-typed* w.r.t. a list of definitions Δ if t is a $\Sigma\mathcal{D}'$-term and for all expressions of the form $(\pi_1(\pi_2^{n-1} d))^{\tau_n}$ occurring in t there is a definition $d =_{def} w : \exists_{x_1:\tau_1}...\exists_{x_n:\tau_n} C$ in Δ. Likewise we define the property of being *well-typed* w.r.t. a list of definitions for $\Sigma\mathcal{D}'$-formulae, $\Sigma\mathcal{D}'$-w^+ and $\Sigma\mathcal{D}'$-w^- proof-witnesses, $\Sigma\mathcal{D}'H$-formulae, $\Sigma\mathcal{D}'G$-formulae and $\Sigma\mathcal{D}'C$-formulae. A list of definitions Δ is *well-formed* w.r.t. a signature Σ if the assertion $\emptyset; \langle\rangle \vdash_{wfld} \Delta$ can be proved by using the inference rules in Fig. 2.

$$\frac{}{\mathcal{D}'; \Delta \vdash_{wfld} \langle\rangle}\ axiom$$

$$\frac{\mathcal{D}' \cup \{d\}; \Delta, d =_{def} w : C \vdash_{wfld} \Delta'}{\mathcal{D}'; \Delta \vdash_{wfld} d =_{def} w : C, \Delta'}$$

The second rule has the following provisos attached:

1. $d \notin \mathcal{D}'$;
2. w is a well-typed $\Sigma\mathcal{D}'$-w^+-proof-witness w.r.t. Δ;
3. C is a well-typed $\Sigma\mathcal{D}'C$-formula w.r.t. Δ.

Fig. 2. The rules for $\vdash_{wfld}$.

We define a new concept of *sequent* [4] as follows: a sequent is a quintuple $\Sigma; \Delta; \Gamma \Rightarrow w : F$, where

1. Σ is a signature;
2. Δ is a well-formed list of definitions w.r.t. Σ and $\mathcal{D}'$ is the set of names being defined in Δ;
3. Γ is a set of the form $w_1 : F_1, ..., w_n : F_n$, where $w_1, ..., w_n$ are well-typed $\Sigma\mathcal{D}'$-w^--proof-witnesses w.r.t. Δ and $F_1, ..., F_n$ are either well-typed $\Sigma\mathcal{D}'H$- or $\Sigma\mathcal{D}'C$-formulae w.r.t. Δ;
4. w is a well-typed $\Sigma\mathcal{D}'$-w^+-proof-witness w.r.t. Δ;
5. F is either a well-typed $\Sigma\mathcal{D}'G$- or $\Sigma\mathcal{D}'C$-formula w.r.t. Δ.

We call $\Sigma; \Delta; \Gamma$ a *basis*.

Let the sequent calculus HH^{cut} be defined by the inference rules of Fig. 3.

Theorem 2. *All proofs in HH^{cut} are uniform proofs.*

Proof. Observe that the only rules that can be applied to non-atomic succedent sequents are right introduction rules. □

[4] We refer to the sequents defined in the previous section by LJ^a sequents.

$$\frac{}{\Sigma;\Delta;\Gamma, w_1^- : A \Rightarrow w_1^- : A}\ axiom$$

$$\frac{\Sigma;\Delta;\pi_1 w_1^- : H_1, \pi_2 w_1^- : H_2, \Gamma \Rightarrow w : A}{\Sigma;\Delta; w_1^- : H_1 \wedge H_2, \Gamma \Rightarrow w : A}\ \wedge\Rightarrow$$

$$\frac{\Sigma;\Delta;\Gamma \Rightarrow w_1 : G_1 \quad \Sigma;\Delta;\Gamma \Rightarrow w_2 : G_2}{\Sigma;\Delta;\Gamma \Rightarrow \langle w_1, w_2\rangle : G_1 \wedge G_2}\ \Rightarrow\wedge$$

$$\frac{\Sigma;\Delta;\Gamma \Rightarrow w : G_1}{\Sigma;\Delta;\Gamma \Rightarrow inl\, w : G_1 \vee G_2}\ \Rightarrow\vee_1 \qquad \frac{\Sigma;\Delta;\Gamma \Rightarrow w : G_2}{\Sigma;\Delta;\Gamma \Rightarrow inr\, w : G_1 \vee G_2}\ \Rightarrow\vee_2$$

$$\frac{\Sigma;\Delta; r : H, \Gamma \Rightarrow w : G}{\Sigma;\Delta;\Gamma \Rightarrow \lambda r.w : H \supset G}\ \Rightarrow\supset$$

$$\frac{\Sigma;\Delta;\Gamma, w_1^- : G \supset A \Rightarrow w_1 : G \quad \Sigma;\Delta;\Gamma, w_1^- : G \supset A, w_1^- w_1 : A \Rightarrow w : A_1}{\Sigma;\Delta;\Gamma, w_1^- : G \supset A \Rightarrow w : A_1}\ \supset\Rightarrow$$

$$\frac{\Sigma;\Delta;\Gamma \Rightarrow w : [t/x]B}{\Sigma;\Delta;\Gamma \Rightarrow \langle t, w\rangle : \exists_{x:\tau} B}\ \Rightarrow\exists \qquad \frac{\Sigma;\Delta;\pi_2 w_1^- : [(\pi_1 w_1^-)^\tau/x]C, \Gamma \Rightarrow w : A}{\Sigma;\Delta; w_1^- : \exists_{x:\tau} C, \Gamma \Rightarrow w : A}\ \exists\Rightarrow$$

$$\frac{\Sigma \cup \{a : \tau\};\Delta;\Gamma \Rightarrow w : [a/x]G}{\Sigma;\Delta;\Gamma \Rightarrow \lambda a.w : \forall_{x:\tau} G}\ \Rightarrow\forall \quad \frac{\Sigma;\Delta; w_1^- t : [t/x]H, w_1^- : \forall_{x:\tau} H, \Gamma \Rightarrow w : A}{\Sigma;\Delta; w_1^- : \forall_{x:\tau} H, \Gamma \Rightarrow w : A}\ \forall\Rightarrow$$

$$\frac{\Sigma;\Delta; d : C, \Gamma \Rightarrow w_1 : A}{\Sigma;\Delta;\Gamma \Rightarrow let\ d = w\ in\ w_1 : A}\ cut \qquad \frac{\Sigma;\Delta; w_1^- : H_1', ..., w_n^- : H_n' \Rightarrow w : A'}{\Sigma;\Delta; w_1^- : H_1, ..., w_n^- : H_n \Rightarrow w : A}\ \lambda conv$$

Provisos:

1. The meta-variables A, A_1, A' range over atomic formulae; the meta-variables G, G_1, G_2 range over G-formulae; the meta-variables $H, H_1, ..., H_n, H_1', ...H_n'$ range over H-formulae; B is either a G-formula or a C-formula and C ranges over C-formulae;
2. in $\Rightarrow\supset$, r is a new abstract realiser;
3. in $\Rightarrow\exists$ and $\forall\Rightarrow$, t is a Σ-term of type τ;
4. in $\Rightarrow\forall$, a is a new parameter;
5. in $\exists\Rightarrow$, w_1^- is of the form $\pi_2(...(\pi_2 d)...)$;
6. in cut, the definition $d =_{def} w : C$ is a member of Δ;
7. in $\lambda conv$, $H_1, ..., H_n, A$ are λ-convertible to $H_1', ..., H_n', A'$ respectively.

Fig. 3. The calculus HH^{cut}.

The inference rules of Fig. 4 define for bases the property of being well-formed: we say that a basis $\Sigma; \Delta; \Gamma$ is *well-formed* if the assertion $\vdash_{wfb} \Sigma; \Delta; \Gamma$ is provable.

$$\frac{}{\vdash_{wfb} \Sigma; \langle\rangle; \Gamma}\ axiom$$

$$\frac{\vdash_{wfb} \Sigma; \Delta; \Gamma \quad \vdash_{HH^{cut}} \Sigma; \Delta; \Gamma \Rightarrow w : C}{\vdash_{wfb} \Sigma; \Delta, d =_{def} w : C; \Gamma}$$

Fig. 4. Inference rules of $\vdash_{wfb}$.

Theorem 3. *Let $\Sigma; \Delta; \Gamma \Rightarrow w : G$ be a sequent where $\Sigma; \Delta; \Gamma$ is a well-formed basis and let Γ be of the form $r_1 : H_1, ..., r_n : H_n$, where $r_1, ..., r_n$ are distinct abstract realisers in $\mathcal{R}$ and $H_1, ..., H_n$ are ΣH-formulae. If $\vdash_{HH^{cut}} \Sigma; \Delta; \Gamma \Rightarrow w : G$ then there is a proof-witness w_1 s.t. $\vdash_{LJ^a} \Sigma; \Gamma^* \Rightarrow w_1 : G^*$, where Γ^*, G^* result from Γ, G respectively by replacing all definienda by their definientia.*

Proof. A sketch of the proof goes as follows. Let p be a proof of $\Sigma; \Delta; \Gamma \Rightarrow w : G$. Then a proof p_1 of $\Sigma; \Gamma^* \Rightarrow w_1 : G^*$ can be built up by following the structure of the proof p_1 and by changing the proof-witnesses accordingly. For example, let

$$\frac{\Sigma; \Delta; \Gamma, d : C \Rightarrow w_3 : A}{\Sigma; \Delta; \Gamma \Rightarrow let\ d = w_2\ in\ w_3 : A}\ cut$$

be an instance of the cut-rule in p. By induction hypothesis we know how to obtain a proof of $\Sigma; \Gamma^* \Rightarrow w_4 : C^*$ from the proof of well-formedness of the basis $\Sigma; \Delta; \Gamma$ and a proof of $\Sigma; \Gamma^*, r : C^* \Rightarrow w_5 : A^*$, where r is a new abstract realiser and Γ^*, C^*,A^* result from Γ,C,A respectively by replacing all definienda by their definientia. Then this instance of the cut-rule in p originates an instance

$$\frac{\Sigma; \Gamma^* \Rightarrow w_4 : C^* \quad \Sigma; \Gamma^*, r : C^* \Rightarrow w_5 : A^*}{\Sigma; \Gamma^* \Rightarrow let\ r = w_4\ in\ w_5 : A^*}\ cut$$

of the cut-rule in p_1. □

Theorem 4. *Let $\Sigma; \Delta; \Gamma \Rightarrow w : G$ be a sequent where $\Sigma; \Delta; \Gamma$ is a well-formed basis. If $\vdash_{HH^{cut}} \Sigma; \Delta; \Gamma \Rightarrow w : G$ then there is a proof-witness w_1 s.t. $\vdash_{HH^{cut}} \Sigma; \langle\rangle; \Gamma^* \Rightarrow w_1 : G^*$, where Γ^*, G^* result from Γ, G respectively* *by replacing all definienda by their definientia; in other words the cut-rule is admissible.*

Proof. Observe that if there is a proof of $\Sigma; \Delta, d =_{def} w : C, \Delta'; \Gamma \Rightarrow w_1 : G$ where there are no applications of the cut-rule with definition $d =_{def} w : C$ then $\Sigma; \Delta, \Delta'^*; \Gamma^* \Rightarrow w_1 : G$ is a provable sequent of HH^{cut}, where Δ'^*, Γ^* result from Δ', Γ respectively by replacing all definienda by their definientia. Then the proof follows by induction on the structure of the proof of $\vdash_{HH^{cut}} \Sigma; \Delta; \Gamma \Rightarrow w : G$ and from the proof of well-formedness of $\Sigma; \Delta; \Gamma$. $\square$

Let Σ be a signature. Let $\mathcal{P}$ be a set of ΣH-formulae $H_1, ..., H_n$. Let Γ be the set containing the pairs $r_1 : H_1, ..., r_n : H_n$, where $r_1, ..., r_n$ are distinct abstract realisers in $\mathcal{R}$. Let Δ be a list of definitions. The pair $\langle \Delta, \mathcal{P} \rangle$ is a *program* if $\Sigma; \Delta; \Gamma$ is a well-formed basis. Assume $\langle \Delta, \mathcal{P} \rangle$ is a program. Assume also that G is a G-formula and $\Sigma; \Delta; \Gamma \Rightarrow W : G$ is a sequent, where W is a dummy. *Achievement* of the goal G w.r.t. the program $\langle \Delta, \mathcal{P} \rangle$ corresponds to a search for a proof of the sequent $\Sigma; \Delta; \Gamma \Rightarrow W : G$ in the calculus HH^{cut}. During the search for a proof of this sequent the proof-witness W is instantiated. If the proof-search is successful W is instantiated with a proof-witness where dummies do not occur. From this proof-witness one can extract, among various other things, the instantiation for the existentially quantified variables occurring in G.

Theorem 5. *Let Σ be a signature and Γ the set containing the pairs $r_1 : H_1, ..., r_n : H_n$, where $r_1, ..., r_n$ are distinct abstract realisers in $\mathcal{R}$ and $H_1, ..., H_n$ are ΣH-formulae. Let G be a ΣG-formula. Then, if $\Sigma; \Gamma \Rightarrow w : G$ is provable in LJ^a then there is a proof-witness w_1 s.t. $\Sigma; \langle\rangle; \Gamma \Rightarrow w_1 : G$ is provable in HH^{cut}.*

Proof. The proof follows from the observations that (i) we only need to consider cut-free proofs of $\Sigma; \Gamma \Rightarrow w : G$; (ii) all LJ^a sequents occurring in the proof of $\Sigma; \Gamma \Rightarrow w : G$ contain only H-formulae in the antecedent and a G-formula in the succedent; (iii) all the LJ^a rules $\wedge \Rightarrow$, $\supset\Rightarrow$, $\forall \Rightarrow$ can be permuted above the LJ^a rules $\Rightarrow \wedge$, $\Rightarrow \vee$, $\Rightarrow\supset$, $\Rightarrow \exists$ and $\Rightarrow \forall$. $\square$

For any provable sequent of HH^{cut} we can find one of its proofs by applying the following search strategy. If the goal formula is not atomic we apply right introduction rules until the goal becomes atomic. When the goal formula is atomic there might be several rules that can be applied. If there is a defined name d occurring in the goal and there is a definition $d =_{def} w : C$ in the list of definitions we apply the cut-rule and we mark this definition as used so that no other applications of the cut-rule are attempted with this definition. We keep applying the cut-rule until no further applications are possible. Next, we apply $\exists\Rightarrow$ until no further applications of $\exists\Rightarrow$ are possible. At this point all the formulae in the antecedent are H-formulae so we proceed by backchaining as usual, i.e. roughly speaking we proceed by breaking up the conjunctions on the left and by unifying the goal formula with the heads of program formulae starting with the formulae that were originated from cut formulae.

4 Example

The example presented below is based on an example given in [Boo84] to show that cut-free proofs may be exponentially longer than proofs using instances of the cut-rule.

Let τ be a primitive type. Let Σ be the signature

$$\{1:\tau, c:\tau\to\tau\to\tau, L:\tau\to o\}.$$

Informally, we can interpret τ as the set of natural numbers and c as the addition of two natural numbers. Let Γ be the set containing only the following pairs:

$$\begin{array}{l} r_1 : L1, \\ r_2 : \forall_{x:\tau}\forall_{x_1:\tau}\forall_{x_2:\tau}(L(c(cxx_1)x_2)\supset L(cx(cx_1x_2))), \\ r_3 : \forall_{x:\tau}(Lx\supset L(cx1)). \end{array}$$

Let $\mathbf{x}, \mathbf{t}$ be vectors of variables and terms respectively, say of size n. Let $\forall\Rightarrow^*$ represent n applications of the rule $\forall\Rightarrow$. Let H,H' be abbreviations for $\forall_{\mathbf{x}}(G\supset A')$, $G\supset A'$ respectively and let r be an abstract realiser. Let Π be the basis $\Sigma;\Delta;\Gamma, r:H$. Let the notation $\Pi, w_1:H_1,...,w_n:H_n$ signify $\Sigma;\Delta;\Gamma, r:H, w_1:H_1,...,w_n:H_n$. Then the rule $\vdash P$ stands for the sequence of inferences shown in Fig. 5.

$$\dfrac{\dfrac{\Pi, r\mathbf{t}:[\mathbf{t}/\mathbf{x}]H'\Rightarrow w_1:[\mathbf{t}/\mathbf{x}]G \qquad \dfrac{}{\Pi,[\mathbf{t}/\mathbf{x}]H', r\mathbf{t}w_1:[\mathbf{t}/\mathbf{x}]A'\Rightarrow r\mathbf{t}w_1:A}\,axiom}{\Pi, r\mathbf{t}:[\mathbf{t}/\mathbf{x}]H'\Rightarrow r\mathbf{t}w_1:A}\supset\Rightarrow}{\Pi\Rightarrow r\mathbf{t}w_1:A}\,\forall\Rightarrow^*$$

Fig. 5. The sequence of inferences $\vdash P$.

Consider we want to prove in LJ^a the goal $L((\lambda x.cx(c11))((\lambda x.cx(c11))1))$ w.r.t. the program Γ using a uniform proof. Figure 6 shows a uniform proof of this goal together with the instantiations of the dummy proof-witnesses. After performing all substitutions, we obtain for the dummy

$$
\begin{array}{cl}
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_7 : L1 & axiom \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_6 : L(c11) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_5 : L(c(c11)1) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_4 : L(c1(c11)) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_3 : L(c(c1(c11))1) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_2 : L(c(c(c1(c11))1)1) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_1 : L(c(c1(c11))(c11)) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_1 : L((\lambda x.cx(c11))(c1(c11))) & \lambda conv \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_1 : L((\lambda x.cx(c11))((\lambda x.cx(c11))1)) & \lambda conv
\end{array}
$$

The instantiations for the dummy proof-witnesses are:
$W_1 \mapsto r_2(c1(c11))11W_2$; $W_2 \mapsto r_3(c(c1(c11))1)W_3$; $W_3 \mapsto r_3(c1(c11))W_4$;
$W_4 \mapsto r_2 111 W_5$; $W_5 \mapsto r_3(c11)W_6$; $W_6 \mapsto r_3 1 W_7$;
$W_7 \mapsto r_1$;

Fig. 6. A uniform proof.

proof-witness W_1 the proof-witness

$$r_2(c1(c11))11(r_3(c(c1(c11))1)(r_3(c1(c11))(r_2 111(r_3(c11)(r_3 1 r_1))))).$$

In this proof the sequence of inferences shown in Fig. 7, for appropriate instantiations of α, occurs twice. If we had used the cut-rule with cut formula $\forall_{x:\tau}(Lx \supset L(cx(c11)))$ we would have needed to prove the sequence of inferences in Fig. 7 only once. We will now show how to construct a proof of this goal using instances of the cut-rule as we explained before.

$$
\begin{array}{cl}
\Sigma\, ;\, \Gamma \Rightarrow W_4 : L\alpha & \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_3 : L(c\alpha 1) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_2 : L(c(c\alpha 1)1) & \vdash P \\
\hline
\Sigma\, ;\, \Gamma \Rightarrow W_1 : L(c\alpha(c11)) & \vdash P
\end{array}
$$

The instantiations for the dummy proof-witnesses are:
$W_1 \mapsto r_2 \alpha 11 W_2$; $W_2 \mapsto r_3(c\alpha 1)W_2$; $W_3 \mapsto r_3 \alpha W_4$.

Fig. 7. A sequence of inferences repeated in the proof of Fig. 6

Let Δ be the list whose only member is the definition

$$+_2 =_{def} \langle \lambda x.cx(c11), \lambda a.\lambda r_4.r_2a11(r_3(ca1)(r_3ar_4))\rangle \\ \qquad\qquad : \exists_{f:\tau\to\tau}\forall_{x:\tau}(Lx \supset L(fx)).$$

A proof of the well-formedness of the basis $\Sigma; \Delta; \Gamma$ can be obtained by combining the axiom $\vdash_{wfb} \Sigma; \langle\rangle; \Gamma$ with the proof presented in Fig. 8. By using the defined name $+_2$ we rewrite the goal $L((\lambda x.cx(c11))((\lambda x.cx(c11))1))$ into $L(\pi_1 +_2 (\pi_1 +_2 1))$, where for brevity we drop the types in $\mathcal{E}$-expressions. Now we show how to construct a proof of the sequent $\Sigma; \Delta; \Gamma \Rightarrow W : L(\pi_1 +_2 (\pi_1 +_2 1))$, where W is a dummy proof-witness, by applying the proof-strategy described in the previous section. The occurrence of the defined name $+_2$ in the atomic goal triggers an instance of the cut-rule with its type being the cut formula to be used. We look up the list of defined names and we find the type $\exists_{f:\tau\to\tau}\forall_{x:\tau}(Lx \supset L(fx))$ for the name $+_2$. We apply the cut-rule and we have now to prove the sequent:

$$(i)\ \ \Sigma; \Delta; +_2 : \exists_{f:\tau\to\tau}\forall_{x\tau}(Lx \supset L(fx)), \Gamma \Rightarrow W_1 : L(\pi_1 +_2 (\pi_1 +_2 1)).$$

As a consequence of applying the cut-rule the proof-witness W is instantiated with

$$let\ +_2 = \langle \lambda x.cx(c11), \lambda a.\lambda r_4.r_2a11(r_3(ca1)(r_3ar_4))\rangle\ in\ W_1.$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma \cup \{a:\tau\}; \langle\rangle; r_4 : La, \Gamma \Rightarrow r_4 : La}\ axiom}{\Sigma \cup \{a:\tau\}; \langle\rangle; r_4 : La, \Gamma \Rightarrow r_3ar_4 : L(ca1)}\ \vdash P}{\Sigma \cup \{a:\tau\}; \langle\rangle; r_4 : La, \Gamma \Rightarrow r_3(ca1)(r_3ar_4) : L(c(ca1)1)}\ \vdash P}{\Sigma \cup \{a:\tau\}; \langle\rangle; r_4 : La, \Gamma \Rightarrow r_2a11(r_3(ca1)(r_3ar_4)) : L(ca(c11))}\ \vdash P}{\Sigma \cup \{a:\tau\}; \langle\rangle; r_4 : La, \Gamma \Rightarrow r_2a11(r_3(ca1)(r_3ar_4)) : L((\lambda x.cx(c11))a)}\ \lambda conv}{\Sigma \cup \{a:\tau\}; \langle\rangle; \Gamma \Rightarrow \lambda r_4.r_2a11(r_3(ca1)(r_3ar_4)) : La \supset L((\lambda x.cx(c11))a)}\ \supset}{\Sigma; \langle\rangle; \Gamma \Rightarrow \lambda a.\lambda r_4.r_2a11(r_3(ca1)(r_3ar_4)) : \forall_{x:\tau}(Lx \supset L((\lambda x.cx(c11))x))}\ \Rightarrow\forall}{\Sigma; \langle\rangle; \Gamma \Rightarrow \langle \lambda x.cx(c11), \lambda a.\lambda r_4.r_2a11(r_3(ca1)(r_3ar_4))\rangle : \exists_{f:\tau\to\tau}\forall_{x:\tau}(Lx \supset L(fx))}\ \Rightarrow\exists$$

Fig. 8. A proof of well-formedness of $\Sigma; \Delta; \Gamma$.

Figure 9 shows a proof of the sequent (i) together with the instantiations of the dummy proof-witnesses generated during the search for the proof. In this proof, we start by replacing in the cut formula the symbol f by the term $\pi_1 +_2$. Then we use the cut formula twice to simplify the goal and we are left with an axiom sequent.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma; \Delta; \pi_2 +_2 : \forall_{x:\tau}(Lx \supset L(\pi_1 +_2 x)), \Gamma \Rightarrow W_3 : L1}\ axiom}{\Sigma; \Delta; \pi_2 +_2 : \forall_{x:\tau}(Lx \supset L(\pi_1 +_2 x)), \Gamma \Rightarrow W_2 : L(\pi_1 +_2 1)}\ \vdash P}{\Sigma; \Delta; \pi_2 +_2 : \forall_{x:\tau}(Lx \supset L(\pi_1 +_2 x)), \Gamma \Rightarrow W_1 : L(\pi_1 +_2 (\pi_1 +_2 1))}\ \vdash P}{\Sigma; \Delta; +_2 : \exists_{f:\tau \to \tau}\forall_{x:\tau}(Lx \supset L(fx)), \Gamma \Rightarrow W_1 : L(\pi_1 +_2 (\pi_1 +_2 1))}\ \exists \Rightarrow$$

The instantiations for the dummy proof-witnesses are:
$W_1 \mapsto \pi_2 +_2 (\pi_1 +_2 1)W_2$; $W_2 \mapsto \pi_2 +_2 1W_3$; $W_3 \mapsto r_1$.

Fig. 9. A proof of (i).

Replacing all the dummy proof-witnesses by their instantiations we obtain for the initial goal $L(\pi_1 +_2 (\pi_1 +_2 1))$ the proof-witness

$$\begin{array}{l} let\ +_2 = \langle \lambda x.cx(c11), \lambda a.\lambda r_4.r_2a11(r_3(ca1)(r_3ar_4))\rangle \\ in\ \pi_2 +_2 (\pi_1 +_2 1(\pi_2 +_2 1r_1)). \end{array}$$

One of the consequences of the Curry-Howard correspondence between types and propositions is the relation between normalisation and cut-elimination. For if we normalise the proof-witness we obtained for the goal $L(\pi_1 +_2 (\pi_1 +_2 1))$ we obtain the term

$$r_2(c1(c11))11(r_3(c(c1(c11))1)(r_3(c1(c11))(r_2111(r_3(c11)(r_31r_1))))),$$

which is a witness for a cut-free proof of $L((\lambda x.cx(c11))(\lambda x.cx(c11))1))$. In fact this proof-witness is the same as the proof-witness we obtained for the uniform proof of Fig. 6.

5 Related and Future Work

The typed logic programming language λProlog [NM88] is based on the logic of higher-order hereditary Harrop formulae for which uniform proofs are complete, as shown in [MNPS91]. It supports modular programming, abstract data types and higher-order functions and predicates. We showed how to extend the logic of first-order hereditary Harrop formulae to have definitions of names to control the applications of the cut-rule. We intend to look at the possibility of having non-hereditary Harrop formulae in a program provided they are paired with a proof-witness that would guide the application of left introduction rules. In fact this problem arises if we try to extend the set of formulae we allowed as cut formulae in the setting of first-order hereditary Harrop formulae.

Two different views of logic programming based on the system of dependent types LF [HHP87] are given in [Pfe91] and in [Pym90].

Elf [Pfe91], [Pfe92] is a logic programming language based on types through the propositions-as-types correspondence. Achieving a goal (type) G w.r.t. a program (context) Γ corresponds to a search for a closed object M of type G, where the language is determined by a signature Σ such that $\Gamma \vdash_\Sigma M : G$ is provable in a natural deduction formulation of LF. The answer to a query is not only a substitution for the free variables but a term of query type. Elf has two sorts of incompleteness w.r.t. LF, one due to the use of a depth-first search and the other caused by the undecidability of unification for the definitional equality $\equiv_{\beta\eta}$ used in LF. As in our work, in Elf computation corresponds to a search for an object of query type, but whereas in our case we search for a proof in a sequent calculus, Elf searches for a natural deduction proof. A major difference between Elf and our work is that in Elf a program is solely a list of type assignments to variables; in our proposal a program is also allowed to contain definitions, where a new variable is introduced as a name for an expression of a certain type.

In [Pym90], [PW91] logic programming is seen as a search for a proof of the sequent $\Gamma(\alpha) \Rightarrow_\Sigma A(\alpha)$, where Σ is a signature determining a language; (α) is a list of indeterminates; Γ is a context assigning types that may have occurrences of the indeterminates in (α) to variables and A is a type that may have occurrences of indeterminates in (α). The result of a successful search is a mapping σ from indeterminates to terms such that there is a term M for which $\Gamma(\alpha)\sigma \vdash_\Sigma M : A(\alpha)\sigma$ is provable in LF. The resulting mapping being what one normally calls answer substitution in logic programming. Although proof-search is carried out in a sequent calculus that allows cut-elimination a search for a proof does not involve uses of the cut-rule.

A Curry-Howard correspondence between a fragment of propositional intuitionistic sequent calculus and a programming language, where evaluation in the programming language corresponds to cut-elimination in the sequent calculus, is presented in [Wad93]. Evaluation in this programming language is different from evaluation in the programming language obtained by composing a translation of sequent calculus into natural deduction with the Curry-Howard correspondence between natural deduction and λ-calculus. In our work term assignment is done through this composition of a translation of sequent calculus into natural deduction with the Curry-Howard correspondence. In future work we intend to use a term assignment similar to the one described in [Wad93] and investigate what different evaluation mechanisms can be obtained from different algorithms to perform cut-elimination.

In [PW92] it is suggested that in logic programming the achievement of a goal w.r.t. a program can be usefully divided into two phases: the first being proof-search and the second being answer extraction. This idea was considered for cut-free systems. In fact the same idea can be applied to systems with cut-rule. For example, the two phases might be (i) the search for a proof-witness of a goal in a system with the cut-rule; (ii) the extraction of all the terms used to replace existentially quantified variables in the goal and subsequent normalisation of these terms, thus avoiding the normalisation of the entire proof-witness.

An analysis of logic programs as types in the sense of the Curry-Howard correspondence is given in [Lip92]. A logic program is transformed into an equational specification over the term model by exploiting a uniformity in the predicates and parameters in the program. A program is written as a realisability goal and there is a search for a function that returns a proof-witness for every choice of parameters. This mechanism of synthesising functions can be seen as a way of generating automatically cut formulae. For it should be possible to employ the idea of using definitions to guide the proof-search by defining names for the synthesised functions.

Deliverables [MB93] are proof-witnesses $\langle f, w\rangle$ of formulae of the form $\exists_{x:\tau_1\rightarrow\ldots\rightarrow\tau_n} F$, where $\tau_1, \ldots, \tau_n$ are primitive types different from o and F is a formula. In [MB93] it is argued that deliverables are the products a software house should deliver to its customers, i.e. a program f and a proof w that the program meets the original specification F. Elsewhere [Pin] we exploit the idea of using definitions to control the applications of the cut-rule to give a proof-theoretic semantics to integrate logic and functional programming by defining names for deliverables.

The language LeFun, as presented in [AKN89], is a programming language that integrates logic and functional programming. In this language a program is a list of logical formulae together with a list of definitions. A definition in LeFun has the form $name =_{def} \lambda\text{-}term$, thus leaving out the specification the λ-term satisfies as well as a proof-witness for that. The computation mechanism is called *residuation*, which is a mechanism to delay unification until the arguments of functions are fully instantiated. In forthcoming work we expect to make precise the relation of LeFun with the proof-theoretic semantics to integrate logic and functional programming based on the idea of using names to guide the proof-search.

6 Conclusions

There are proofs that can be exponentially shorter if they are allowed to use the cut-rule. The problem of automating proof-search in a calculus with a cut-rule is that we may apply the cut-rule to any sequent and once we have decided to apply the cut-rule we still have the freedom of applying the cut-rule with any formula as cut formula. Our idea of using definitions to guide the proof-search restricts the cut-rule in such a way that its application is allowed only in case there is a defined name occurring in the goal formula, and in this case we only attempt the cut-rule with the type of the defined name as cut formula. Thus, we can have a goal-directed proof-search that in some cases will find proofs exponentially shorter than we would find with a cut-free search procedure.

References

[AKN89] H. Ait-Kaci and R. Nasr. Integrating logic and functional programming. *Lisp and Symbolic Computation*, 2:51–89, 1989.

[Bee89] M. Beeson. Some applications of Gentzen's proof theory in automated deduction. In P. Schroeder-Heister, editor, *Extensions of Logic Programming, international workshop, Tübingen, 1989, proceedings*, volume 475 of *LNCS*, pages 101–156. Springer-Verlag, 1989.

[Boo84] G. Boolos. Don't eliminate cut. *Journal of Philosophical Logic*, 13:373–378, 1984.

[Gen69] G. Gentzen. Investigations into logical deduction. In M. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1969.

[GLT89] J.Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proc. Second Annual Symposium on Logic in Computer Science*, pages 194–204. IEEE, 1987.

[Lip92] J. Lipton. Relating logic programming and propositions-as-types: a logical compilation. In *Proc. of the Workshop on Types for Proofs and Programs, Båstad, Sweden*, 1992.

[MB93] J. McKinna and R. Burstall. Deliverables: a categorical approach to program development in type theory. In A. Borzyszkowski and S. Sokolowski, editors, *Mathematical Foundations of Computer Science 1993*, volume 711 of *LNCS*, pages 32–67. Springer-Verlag, 1993.

[Mil90] D. Miller. Abstractions in logic programming. In P. Odifreddi, editor, *Logic and Computer Science*, pages 329–359. Academic Press, 1990.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Napoli, 1984.

[MNPS91] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[NM88] G. Nadathur and D. Miller. An overview of λProlog. In *Proc. Fifth Internat. Logic Programming Conference, Seattle*, pages 810–827. MIT Press, 1988.

[Pfe91] F. Pfenning. Logic programming in the LF logical framework. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe92] F. Pfenning. Dependent types in logic programming. In F. Pfenning, editor, *Types in Logic Programming*, chapter 10, pages 285–311. MIT, 1992.

[Pin] L. Pinto. Proof-theoretic semantics and integration of logic and functional programming, (in preparation).

[PW91] D. Pym and L. Wallen. Proof-search in the $\lambda\Pi$-calculus. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 309–340. Cambridge University Press, 1991.

[PW92] D. Pym and L. Wallen. Logic programming via proof-valued computations. In K. Broda, editor, *Proc. 4th UK Conf. on Logic Programming, London, 1992*. Springer, 92.

[Pym90] D. Pym. *Proofs, search and computation in general logic.* PhD thesis, University of Edinburgh, 1990.

[Tho91] S. Thompson. *Type Theory and Functional Programming.* Addison-Wesley, 1991.

[Wad93] P. Wadler. A Curry-Howard isomorphism for sequent calculus. Preprint, University of Glasgow, December 1993.

This article was processed using the LaTeX macro package with LLNCS style