A Formal Framework for Synthesis and Verification of Logic Programs

Alessandro Avellone, Mauro Ferrari, and Camillo Fiorentini

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
via Comelico 39, 20135 Milano, Italy
{avellone,ferram,fiorenti}@dsi.unimi.it

Abstract. In this paper we present a formal framework, based on the notion of *extraction calculus*, which has been applied to define procedures for extracting information from constructive proofs. Here we apply such a mechanism to give a proof-theoretic account of SLD-derivations. We show how proofs of suitable constructive systems can be used in the context of deductive synthesis of logic programs, and we state a link between constructive and deductive program synthesis.

1 Introduction

It is well known that formal proofs can be used for program synthesis and program verification, and this essentially depends on the availability of an *information extraction mechanism* allowing the capture in an *uniform way* of the implicit algorithmic content of a proof. In this paper we present a formal framework, based on the notion of *extraction calculus*, which has been devised by the authors [4,5,6,7,8] and applied to define procedures for extracting information from proofs of a great variety of logical systems.

Here we apply extraction calculi to give a proof-theoretic account of SLD-derivations. We show how proofs of suitable constructive systems can be used in the context of deductive synthesis of logic programs, and we state a link between constructive and deductive program synthesis (see [3] for a survey on the various approaches to logic program synthesis). We will consider *extended logic programs* constituted by *extended program clauses*, a generalization of usual program clauses where negated atomic formulas are allowed also in the head. Since the choice of the semantics for extended logic programs is problematic in the usual semantical paradigms of Logic Programming, we develop our approach in the setting of *specification frameworks* developed in [9,10]. The intended semantics of specification frameworks is given by *isoinitial models*, a semantics that can be fruitfully combined with a constructive proof theory (see [12] for a comprehensive discussion on isoinitial models, Abstract Data Types Specification and constructive proof-theory).

The main result of our paper concerns the extraction of extended logic programs from proofs in a natural deduction calculus. In particular we prove that the logic programs consisting of the extended program clauses occurring in a

Kung-Kiu Lau (Ed.): LOPSTR 2000, LNCS 2042, pp. 1–17, 2001.

natural deduction proof of a formula $\exists x G(x) \vee \neg \exists x G(x)$, with G atomic, allows us to correctly compute the goal $\leftarrow G(x)$ (i.e., find a term t such that G(t) holds, if such a t exists); here the notion of correctness is the one referred to specification frameworks of [9,10]. This result can be used both to define a synthesis method and to study properties of the deductive synthesis process. Finally we discuss a simple example showing in which sense our framework allows us to treat modularity.

This paper is a first step in the application of extraction calculi to logic program synthesis; we believe that our approach can be further developed and applied to open frameworks and to "disjunctive Logic Programming".

2 Extraction Calculi

In this section we provide a short presentation of our mechanism to extract information from proofs; for a complete discussion see [5,6,7,8]. Our extraction mechanism is based on an abstract definition of the notions of proof and calculus allowing us to treat extraction from Gentzen, Tableau or Hilbert style calculi.

Here we consider first-order languages $\mathcal{L}_\Sigma$ over a set of non logical symbols Σ and the set of logical symbols $\{\wedge, \vee, \rightarrow, \neg, \forall, \exists\}$. A (single-conclusion) *sequent* is an expression $\Gamma \vdash A$, where A is a formula and Γ is a finite set of formulas; we simply write $\vdash A$ to denote a sequent with Γ empty. A *proof* on $\mathcal{L}_\Sigma$ is any finite object π such that:

1. the (finite) set of formulas of $\mathcal{L}_\Sigma$ occurring in π is uniquely determined and nonempty;
2. π proves a sequent $\Gamma \vdash A$, where Γ (possibly empty) is the set of *assumptions* of π, while A is the *consequence* of π.

We use the notation $\pi : \Gamma \vdash A$ to mean that $\Gamma \vdash A$ is the sequent proved by π.

In the following a great care will be devoted to bound the logical complexity (*degree*) of the formulas involved in the information extraction process from a proof. The degree $\mathrm{dg}(A)$ of a formula is defined as follows: $\mathrm{dg}(A) = 1$ if A is atomic, $\mathrm{dg}(\neg A) = \mathrm{dg}(A) + 1$; $\mathrm{dg}(A) = \max\{\mathrm{dg}(B), \mathrm{dg}(C)\} + 1$ if A is $B \wedge C$, $B \vee C$ or $B \rightarrow C$; $\mathrm{dg}(A) = \mathrm{dg}(B) + 1$ if A is either $\exists x B(x)$ or $\forall x B(x)$. The degree $\mathrm{dg}(\pi)$ of a proof π is the maximum among the degrees of the formulas occurring in π.

A *calculus* on $\mathcal{L}_\Sigma$ is a pair $(\mathbf{C}, [\cdot])$, where $\mathbf{C}$ is a recursive set of proofs on the language $\mathcal{L}_\Sigma$ and $[\cdot]$ is a recursive map associating with every proof of the calculus the set of its *relevant* subproofs. We require $[\cdot]$ to satisfy the following natural conditions:

1. $\pi \in [\pi]$;
2. for every $\pi' \in [\pi]$, $[\pi'] \subseteq [\pi]$;
3. for every $\pi' \in [\pi]$, $\mathrm{dg}(\pi') \leq \mathrm{dg}(\pi)$.

We remark that any usual single conclusion inference system is a calculus according to our definition. In particular the natural deduction calculi we use in this paper meet this characterization.

Given $\Pi \subseteq \mathbf{C}$, $\mathrm{Seq}(\Pi) = \{\Gamma \vdash A \mid \pi : \Gamma \vdash A \in \Pi\}$ is the set of the *sequents proved in* Π; $\mathrm{Theo}(\Pi) = \{A \mid \ \vdash A \in \mathrm{Seq}(\Pi)\}$ is the set of *theorems proved in* Π, and $[\Pi] = \{\pi' \mid$ there exists $\pi \in \Pi$ such that $\pi' \in [\pi]\}$ is the *closure under subproofs* of Π in the calculus $\mathbf{C}$. Now, let $\mathcal{R}$ be a set of inference rules of the kind

$$\frac{\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n}{\Delta \vdash B}\mathrm{R}$$

(where R is the name of the rule). $\mathcal{R}$ is a set of *extraction rules for* $\mathbf{C}$ (*e-rules* for short) if there exists a function $\phi : \mathbf{N} \to \mathbf{N}$ such that, for every $R \in \mathcal{R}$:

1. R can be uniformly simulated in $\mathbf{C}$ w.r.t. ϕ. That is: for every $\pi_1,\ldots,\pi_n$ in $\mathbf{C}$ proving the premises of the rule R, there exists a proof π of the consequence of R such that $\mathrm{dg}(\pi) \leq \max\{\phi(\mathrm{dg}(\pi_1)), \ldots, \phi(\mathrm{dg}(\pi_n))\}$.
2. R is *non-increasing*. That is: the degree of every formula occurring in $\Delta \vdash B$ is bounded by the degree of a formula occurring in $\Gamma_1 \vdash A_1, \ldots, \Gamma_n \vdash A_n$.

Condition (1) says that an extraction rule must be an admissible rule for $\mathbf{C}$, and must be simulated in a uniform way (w.r.t. the degrees) in the calculus $\mathbf{C}$. This means that an extraction rule respects the deductive power of the original calculus in a uniform way. On the other hand Condition (2) says that an extraction rule can coordinate or decompose the information contained in the premises, but it must not create "new" information. Here, Condition (2) accomplishes this by imposing a bound on the degree of the consequence, but more sophisticated conditions using the subformula property or a goal oriented strategy can be used (see [5]).

Examples of e-rules for the natural deduction calculus for Intuitionistic Logic are the *cut rule* and the *substitution rule*

$$\frac{\Gamma \vdash H \quad \Gamma, H \vdash A}{\Gamma \vdash A}\mathrm{CUT}$$

$$\frac{\Gamma \vdash A}{\theta\Gamma \vdash \theta A}\mathrm{SUBST} \qquad \text{with } \theta \text{ a substitution}$$

In our treatment a central role is played by the e-rule SLD^+ introduced in Section 5.

Now, given a set $\mathcal{R}$ of e-rules for $\mathbf{C}$ and $\Pi \subseteq \mathbf{C}$, the *extraction calculus for* Π, denoted by $\mathbb{ID}(\mathcal{R}, [\Pi])$, is defined as follows:

1. If $\Gamma \vdash A \in \mathrm{Seq}([\Pi])$, then $\tau \equiv \Gamma \vdash A$ is a proof-tree of $\mathbb{ID}(\mathcal{R}, [\Pi])$.
2. If $\tau_1 : \Gamma_1 \vdash A_1, \ldots, \tau_n : \Gamma_n \vdash A_n$ are proof-trees of $\mathbb{ID}(\mathcal{R}, [\Pi])$ and

$$\frac{\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n}{\Delta \vdash B}\mathrm{R}$$

is a rule of $\mathcal{R}$, then the proof-tree

$$\tau \equiv \frac{\tau_1 : \Gamma_1 \vdash A_1 \ \ldots \ \tau_n : \Gamma_n \vdash A_n}{\Delta \vdash B}\mathrm{R}$$

with root $\Delta \vdash B$ belongs to $\mathbb{ID}(\mathcal{R}, [\Pi])$.

The following properties follow from the definition of e-rule (see [4,8]):

Theorem 1. *Let $\mathcal{R}$ be a set of e-rules for $\mathbf{C}$ and let Π be a recursive subset of $\mathbf{C}$ with $\mathrm{dg}(\Pi) \leq c$ ($c \geq 1$). Then:*

1. *There exists $h \geq 0$ (which depends on $\mathcal{R}$ and c) such that $\mathrm{dg}(\tau) \leq h$ for every $\tau \in \mathbb{ID}(\mathcal{R}, [\Pi])$.*
2. *There exist a subset $\Pi' \subseteq \mathbf{C}$ and $k \geq 0$ (which depends on $\mathcal{R}$ and c) such that $\mathrm{dg}(\Pi') \leq k$ and $\mathrm{Seq}(\Pi') = \mathrm{Seq}(\mathbb{ID}(\mathcal{R}, [\Pi]))$.*

A set of proofs Π is *constructive* if it meets the *disjunction property* (DP):

(DP) $A \vee B \in \mathrm{Theo}(\Pi)$ implies $A \in \mathrm{Theo}(\Pi)$ or $B \in \mathrm{Theo}(\Pi)$;

and the *explicit definability property* (ED):

(ED) $\exists x A(x) \in \mathrm{Theo}(\Pi)$, implies $A(t/x) \in \mathrm{Theo}(\Pi)$ for some ground term t of the language.

Definition 1. *A calculus $\mathbf{C}$ is* uniformly constructive *if there exists a set of e-rules $\mathcal{R}$ for $\mathbf{C}$ such that, for every recursive subset Π of $\mathbf{C}$, $\mathrm{Theo}(\mathbb{ID}(\mathcal{R}, [\Pi]))$ is constructive.*

The properties of a uniformly constructive calculus $\mathbf{C}$ assure that, if $\pi : \vdash \exists x A(x) \in \mathbf{C}$, then we can determine a ground term t such that $A(t)$ is provable in $\mathbf{C}$ exploiting the information contained in the proof π by means of the calculus $\mathbb{ID}(\mathcal{R}, [\pi])$. Moreover, such information can be searched in the calculus $\mathbb{ID}(\mathcal{R}, [\pi])$ by means of an enumerative procedure only involving formulas of bounded logical complexity (by Point (1) of Theorem 1). This allows us to define procedures for extracting information from constructive proofs which are of interest in the fields of program synthesis and formal verification [1,6].

In [6,7,8] a wide family of constructive systems (involving theories formalizing Abstract Data Types) is shown to be uniformly constructive, while in [8] an example of a constructive but not uniformly constructive formal system is provided. However, the proofs of uniform constructivity of all the systems discussed in [6,8] rely on extraction rules which are not suitable to get efficient proof search in $\mathbb{ID}(\mathcal{R}, [\Pi])$; in particular, all these results involve the cut rule (for a discussion on the complexity of these extraction procedures see [2]). On the other hand, in [5] it is proven that $\mathbb{ID}(\mathcal{R}, [\Pi])$ can be characterized as a goal-oriented calculus for suitable proofs involving Hereditary Harrop Formulae (see [13]). In Section 5, we show that the SLD$^+$ rule is sufficient to decide the explicit definability property and the disjunction property for suitable constructive proofs only involving *program clauses* as assumptions; this immediately yields a logic program synthesis method.

3 Specification Frameworks for Logic Programs

Specification frameworks, introduced in [9,10], establish the semantical setting for logic programs. A *framework* $\mathcal{F}$ is a theory in some language $\mathcal{L}_\Sigma$ (namely, a recursively enumerable set of closed formulas of $\mathcal{L}_\Sigma$) and it must define in an unambiguous way the intended semantics of a logic program. In the tradition of Logic Programming [11], the semantics of a program P is determined by P itself; indeed, the *initial* model of P is assumed to be the intended model of P. On the other hand, if we enlarge the language of logic programs some problems arise; for instance, when we introduce negation, it is not clear what the semantics of P should be. We aim to use a generalization of program clauses (we call *extended program clauses*), where negated atomic formulas are allowed also in the head. In this case, the choice of the semantics of P is rather problematic, and even the semantics of completion does not work. To overcome these difficulties, we separate the task of defining the semantics from the task of synthesizing logic programs. Firstly we introduce a framework $\mathcal{F}$ to describe the domain of the problem, exploiting the expressiveness of first-order languages; within the framework we explicitly define the relations to be computed and the goals to be solved by means of a *specification* $\langle D_r, \mathcal{G}_r \rangle$. The link between frameworks and logic programs is settled by natural calculi. We show how the problem of compute a goal G in $\mathcal{F}$ can be reduced to the problem of finding a proof of $\exists x G(x) \vee \neg \exists x G(x)$ in the natural calculus.

In our formalization we aim to exploit the *full* first-order language, without imposing restrictions on the form of the axioms of $\mathcal{F}$; in such a general setting, initial semantics is inadequate, thus, following [9,10], we use the *isoinitial semantics*. We briefly recall some definitions. We say that $\mathfrak{I}$ is an *isoinitial* model of a theory $\mathcal{F}$ if, for every model $\mathfrak{M}$ of $\mathcal{F}$, there exists a unique *isomorphic embedding* (i.e., an homomorphism which preserves relations and their negations) of $\mathfrak{I}$ in $\mathfrak{M}$ (if we take homomorphisms, we obtain the definition of initial models). We stress that an isoinitial model of a theory $\mathcal{F}$ (if it exists) is unique up to isomorphisms, thus it can be considered the intended semantics of $\mathcal{F}$. We recall that a model is *reachable* if every element of its domain is denoted by a ground term of the language; moreover, a theory $\mathcal{F}$ is *atomically complete* if, for every atomic closed formula A of $\mathcal{L}_\Sigma$, either A or $\neg A$ is a logical consequence of $\mathcal{F}$. We can characterize theories which admit isoinitial models as follows:

Theorem 2. *If a theory $\mathcal{F}$ has at least one reachable model, then $\mathcal{F}$ admits an isoinitial model if and only if it is atomically complete.*

This yields the definition of closed framework (see [9,10]):

Definition 2. *A* closed *framework with language $\mathcal{L}_\Sigma$ is a theory $\mathcal{F}$ which satisfies the following properties:*

1. **Reachability**. *There is at least one model of $\mathcal{F}$ reachable by a subset $\mathcal{C}$ of the constant and function symbols of $\mathcal{L}_\Sigma$, called the* construction symbols. *The ground terms containing only construction symbols will be called* constructions.

2. **Freeness**. *$\mathcal{F}$ proves the* freeness axioms *[16] for the construction symbols.*
3. **Atomic completeness**. *$\mathcal{F}$ is atomically complete.*

For instance, Peano Arithmetic $\mathcal{PA}$ is a closed framework: indeed, $\mathcal{PA}$ is atomically complete, the construction symbols are the constant '0' and the unary function symbol 's', the constructions are the ground terms 0, $s(0)$, $s(s(0))\ldots$. Note that $+$ and $*$ are not construction symbols, since they do not satisfy freeness axioms. Given a closed framework $\mathcal{F}$, we can build a particular isoinitial model $\mathfrak{I}$ of $\mathcal{F}$, we call it the *canonical model* of $\mathcal{F}$, in the following way:

(C1) The domain of $\mathfrak{I}$ is the set of constructions.
(C2) A function f is interpreted in $\mathfrak{I}$ as the function $f_{\mathfrak{I}}$ defined as follows: for every tuple $\underline{t}$ of constructions, the value of $f_{\mathfrak{I}}(\underline{t})$ is the construction s such that $\mathcal{F} \models f(\underline{t}) = s$.
(C3) Every relation symbol r is interpreted in $\mathfrak{I}$ as the relation $r_{\mathfrak{I}}$ such that, for every tuple $\underline{t}$ of constructions, $\underline{t}$ belongs to $r_{\mathfrak{I}}$ if and only if $\mathcal{F} \models r(\underline{t})$.

By atomic completeness, canonical models are representative of any other model as regards the validity of the ground atomic or negated ground atomic formulas; more precisely, for every relation r and every tuple $\underline{t}$, $\mathfrak{I} \models r(\underline{t})$ iff, for every model $\mathfrak{M}$ of $\mathcal{F}$, $\mathfrak{M} \models r(\underline{t})$, iff $\mathcal{F} \models r(\underline{t})$.

Within a framework, a goal is represented by a formula $\exists y\, R(t, y)$, where R is any formula of $\mathcal{L}_\Sigma$, to be computationally understood as "find a term s such that $R(t, s)$ is valid in the framework $\mathcal{F}$". Since goals must be handled by logic programs, we need to introduce a new binary relation symbol r which is equivalent in $\mathcal{F}$ to the formula R.

Definition 3. *A* specification *in a framework $\mathcal{F}$ with language $\mathcal{L}_\Sigma$ consists of:*

1. *A definition axiom $D_r \equiv \forall \underline{x} \forall \underline{y}(r(\underline{x}, \underline{y}) \leftrightarrow R(\underline{x}, \underline{y}))$ where $R(\underline{x}, \underline{y})$ is a first-order formula in the language $\mathcal{L}_\Sigma$;*
2. *A set $\mathcal{G}_r$ of goals of the form $\exists \underline{y}\ r(\underline{t}_0, \underline{y}), \exists \underline{y}\ r(\underline{t}_1, \underline{y}), \ldots$, where the $\underline{t}_k$'s are tuples of constructions.*

We can expand in a natural way the canonical model $\mathfrak{I}$ of $\mathcal{F}$ to the language $\mathcal{L}_{\Sigma \cup \{r\}}$ still preserving the semantics of the symbols in $\mathcal{L}_\Sigma$. However, it may happen that the expanded model $\mathfrak{I}_r$ is not even a canonical model for $\mathcal{F} \cup \{D_r\}$, since Condition $(C3)$ might not hold; therefore the model $\mathfrak{I}_r$ is not representative as regards the relation r. To avoid this, we only consider axioms D_r which define decidable relations, according to the following definition.

Definition 4. *Let $\mathcal{F}$ be a closed framework and let D_r be the definition axiom $\forall \underline{x} \forall \underline{y}(r(\underline{x}, \underline{y}) \leftrightarrow R(\underline{x}, \underline{y}))$. We say that D_r completely defines r in $\mathcal{F}$ iff $\mathcal{F} \cup D_r$ satisfies the atomic completeness property.*

If D_r completely defines r in $\mathcal{F}$, then also $\mathcal{F} \cup D_r$ is a closed framework and the model $\mathfrak{I}_r$ which expands $\mathfrak{I}$ is actually the canonical model of $\mathcal{F} \cup D_r$. Clearly, a relation r satisfying Definition 3 may be already in $\mathcal{L}_\Sigma$; in this case there is no need of expanding $\mathcal{L}_\Sigma$ and $\mathcal{F} \cup D_r$ is trivially atomically complete.

Finally, we introduce extended logic programs. An *extended program clause* is any formula of the kind $A_1 \wedge \ldots \wedge A_n \rightarrow B$, where $n \geq 0$ and $A_1, \ldots, A_n, B$ are atomic formulas or negated atomic formulas; this generalizes the notion of Horn clause (where all formulas are positive) and of program clause (where B must be positive) explained in [11]. An *extended logic program* is a finite set of extended program clauses. As the inference rule, we use the rule SLD$^+$ defined in Section 5. The notion of *answer substitution* for extended logic programs is the same given for logic programs, that is: given an extended logic program P, a goal $\exists \underline{y}\ r(\underline{t}, \underline{y})$ and a substitution θ, θ is a correct answer substitution for $P \cup \{\leftarrow r(\underline{t}, \underline{y})\}$ if $\theta r(\underline{t}, \underline{y})$ is a logical consequence of P.

The link between specifications and programs is stated by the following definitions.

Definition 5. *Let $\langle D_r, \mathcal{G}_r \rangle$ be a specification in a framework $\mathcal{F}$. A program P is* totally correct in a model $\mathfrak{M}$ of $\mathcal{F}$ w.r.t. $\langle D_r, \mathcal{G}_r \rangle$ *if, for every $\exists \underline{y}\ r(\underline{t}, \underline{y}) \in \mathcal{G}_r$, it holds that:*

1. *If $\exists \underline{y}\ r(\underline{t}, \underline{y})$ is valid in $\mathfrak{M}$, then there is at least a computed answer substitution for $P \cup \{\leftarrow r(\underline{t}, \underline{y})\}$;*
2. *If σ is a computed answer substitution for $P \cup \{\leftarrow r(\underline{t}, \underline{y})\}$, then $\sigma r(\underline{t}, \underline{y})$ is valid in $\mathfrak{M}$.*

In the sequel we consider specifications which completely define new relations, thus total correctness is with respect to the canonical model of $\mathcal{F} \cup \mathcal{D}_r$ as stated in the following definition.

Definition 6. *Let $\langle D_r, \mathcal{G}_r \rangle$ be a specification in a closed framework $\mathcal{F}$ and let D_r completely define r in $\mathcal{F}$. A program P is* totally correct in $\mathcal{F} \cup D_r$ *iff P is totally correct in the canonical model $\mathfrak{I}_r$ of $\mathcal{F} \cup D_r$.*

4 Natural Deduction Calculi for Program Extraction

As a basis of our synthesis process we will use the calculus $\mathcal{ND}_{\mathcal{C}}(\mathcal{T})$ obtained by adding to the natural calculus $\mathcal{ND}$ of Table 1 the rule $I_{\mathcal{T}}$ to introduce as axioms the formulas of a theory $\mathcal{T}$, and the induction rule Ind$^{\mathcal{C}}$ associated with a finite set $\mathcal{C}$ of constant and function symbols.

We point out that introduction/elimination rules for $\wedge, \vee, \rightarrow, \exists, \forall$ in Table 1 are the usual ones for Minimal Logic (see, e.g., [15]), while we use special rules for negation. In the rules of the caclulus we use square brackets to denote the assumptions discharged by the rule application.

Let $\mathcal{T}$ be a theory; the rule $I_{\mathcal{T}}$ is as follows:

$$\frac{}{A}\, I_{\mathcal{T}} \quad \text{with } A \in \mathcal{T}$$

Let us assume $\mathcal{C}$ to contain the constant and function symbols $s_1, \ldots, s_n$; the *Cover Set Induction Rule* Ind$^{\mathcal{C}}$ associated with $\mathcal{C}$ is the rule

Table 1. The calculus $\mathcal{ND}$

$$\frac{\begin{array}{cc}\Gamma_1 & \Gamma_2\\ \vdots\,\pi_1 & \vdots\,\pi_2\\ A & B\end{array}}{A\wedge B}\wedge I \qquad \frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ A\wedge B\end{array}}{A}\wedge E_l \qquad \frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ A\wedge B\end{array}}{B}\wedge E_r$$

$$\frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ A\end{array}}{A\vee B}\vee I_l \qquad \frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ B\end{array}}{A\vee B}\vee I_r \qquad \frac{\begin{array}{ccc}\Gamma_1 & \Gamma_2,[A] & \Gamma_3,[B]\\ \vdots\,\pi_1 & \vdots\,\pi_2 & \vdots\,\pi_3\\ A\vee B & C & C\end{array}}{C}\vee E$$

$$\frac{\begin{array}{c}\Gamma,[A]\\ \vdots\,\pi\\ B\end{array}}{A\to B}\to I \qquad \frac{\begin{array}{cc}\Gamma_1 & \Gamma_2\\ \vdots\,\pi_1 & \vdots\,\pi_2\\ A & A\to B\end{array}}{B}\to E$$

$$\frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ A(p)\end{array}}{\forall x\ A(x)}\forall I \quad \text{where } p \text{ does not occur free in } \Gamma\ ^* \qquad \frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ \forall x\ A(x)\end{array}}{A(t)}\forall E$$

$$\frac{\begin{array}{c}\Gamma\\ \vdots\,\pi\\ A(t)\end{array}}{\exists x\ A(x)}\exists I \qquad \frac{\begin{array}{cc}\Gamma_1 & \Gamma_2,[A(p)]\\ \vdots\,\pi_1 & \vdots\,\pi_2\\ \exists x\ A(x) & B\end{array}}{B}\exists E \quad \text{where } p \text{ does not occur free in } \Gamma_2,\ \exists x A(x) \text{ or } B\ ^*$$

$$\frac{\begin{array}{cc}\Gamma_1 & \Gamma_2\\ \vdots\,\pi_1 & \vdots\,\pi_2\\ A & \neg A\end{array}}{B}\ Contr \qquad \frac{\begin{array}{c}[H_1,\ldots,H_n,A]\\ \vdots\,\pi\\ B\wedge\neg B\end{array}}{H_1\wedge\ldots\wedge H_n\to\neg A}I\neg$$

with $H_1,\ldots,H_n$ atomic or negated formulas. H_1, $\ldots$, H_n, A are the only undischarged assumptions of π.

* p is the *proper parameter* of the rule.

$$\frac{\begin{array}{ccc}\Gamma_1,[\Delta_1] & & \Gamma_n,[\Delta_n]\\ \vdots\,\pi_1 & & \vdots\,\pi_n\\ A(t_1) & \cdots & A(t_n)\end{array}}{A(x)}\ \mathrm{Ind}^{\mathcal{C}}$$

where x is any variable and, for $1 \leq j \leq n$:

1. If s_j is a constant symbol then Δ_j is empty and t_j coincides with s_j;
2. If s_j is a function symbol of arity k, then $\Delta_j = \{A(y_1), \ldots, A(y_k)\}$ and t_j is the term $s_j(y_1, \ldots, y_k)$.

The formulas in Δ_j are the induction hypotheses, the variables $y_1, \ldots, y_k$ are called *proper parameters* of the Cover Set Induction Rule and must not occur free in $\Gamma_1, \ldots, \Gamma_n, A(x)$; we extend to the rule $\text{Ind}^{\mathcal{C}}$ the conditions on proper parameters made in [15,17]. Finally, the induction hypothesis in the Δ_j's are discharged by the rule application.

We say that a proof π of $\mathcal{ND}_{\mathcal{C}}(\mathcal{T})$ proves the sequent $\Gamma \vdash A$ ($\pi : \Gamma \vdash A$) if Γ is the set of the assumptions on which π depends and A is the end-formula of π; a formula A is *provable in* $\mathcal{ND}_{\mathcal{C}}(\mathcal{T})$ if this calculus contains a proof π of the sequent $\vdash A$ (i.e., π has no undischarged assumptions). Finally, we denote with $\text{depth}(\pi)$ the depth of a natural deduction proof, for a formal definition see [17].

5 Program Extraction from Deductive Proofs

In this section we show how proofs of the calculus $\mathcal{ND}_{\mathcal{C}}(\mathcal{T})$ can be used in the context of deductive synthesis of logic programs (for a general discussion on logic program synthesis see [3]).

Definition 7. *Let Π be a finite set of proofs of $\mathcal{ND}_{\mathcal{C}}(\mathcal{T})$. The* (extended logic) program extracted from Π *is the set $\mathcal{P}_\Pi$ of extended program clauses occurring in* $\text{Theo}([\Pi])$.

Accordingly, the program $\mathcal{P}_\Pi$ extracted from Π contains all the extended program clauses H such that the sequent $\vdash H$ is proven in a subproof of some proof in Π. If Π only consists of the proof π, we say that $\mathcal{P}_\Pi$ is the program extracted from π. To study the proof theoretical properties of the program extracted from Π, we use the extraction calculus $\mathbb{ID}(\{\text{SLD}^+, \text{SUBST}\}, [\Pi])$ ($\mathbb{ID}(\Pi)$ for short), which uses the e-rule SLD^+:

$$\frac{\vdash \theta H_1 \quad \ldots \quad \vdash \theta H_n \qquad \overline{\vdash H_1 \wedge \ldots \wedge H_n \rightarrow K}}{\vdash \theta K}\,\text{SLD}^+$$

where $H_1 \wedge \ldots \wedge H_n \rightarrow K$ is an extended program clause and θ is an arbitrary substitution. If $H_1 \wedge \ldots \wedge H_n \rightarrow K$ is a Horn clause, we obtain the usual SLD rule of Logic Programming; thus, SLD^+ is an extension of SLD where negated formulas are treated as atoms and "negation as failure" is not considered. According to this rule, the proof of the goal θK consists in solving the goals $\theta H_1, \ldots, \theta H_n$ obtained by unifying θK with the program clause $H_1 \wedge \ldots \wedge H_n \rightarrow K$. We remark that θ is not required to be a *most general unifier*, but this does not affect our treatment because any SLD^+ proof using arbitrary substitutions is an instance of a SLD^+ proof using mgu's. For a deeper discussion about the role of the rule SLD^+ in Logic Programming we refer the reader to [14], where a similar rule is studied.

As for the extraction calculus $\mathbb{ID}(\Pi)$, we stress that:

1. the axioms of $\mathbb{ID}(\Pi)$ are the sequents of the form $\vdash A$ occurring in the subproofs of Π;
2. the only applied rules are SLD^+ and SUBST.

We point out that $\mathbb{ID}(\Pi)$ is a "Logic Programming oriented calculus", since a proof $\tau : \ \vdash G(x)$ in $\mathbb{ID}(\Pi)$ can be seen as a refutation of $\mathcal{P}_\Pi \cup \{\leftarrow G(x)\}$, where $\mathcal{P}_\Pi$ is the program extracted from Π. The remarkable point is that, to build the proof τ in $\mathbb{ID}(\Pi)$, not all the information contained in the proofs Π is used, but *only* the sequents of $[\Pi]$ of the form $\vdash H$, with H an extended program clause. This means that the program extracted from the proofs in Π contains all the information needed to compute the goal; moreover, the proofs in Π also contain the information needed to guarantee the correctness of the extracted program as we show in Theorem 3 below.

Finally, we point out that, under some conditions, $\mathbb{ID}(\Pi)$ is constructive. This fact has a relevant consequence in the relationship between proofs of the natural calculus $\mathcal{ND}_\mathcal{C}(\mathcal{T})$ and the deductive power of the extracted logic programs. Suppose there is a proof $\pi : \ \vdash \exists x G(x)$ in $\mathcal{ND}_\mathcal{C}(\mathcal{T})$ and let us assume that the extraction calculus $\mathbb{ID}(\Pi)$ generated by the set $\Pi = \{\pi\}$ is constructive. By definition, $\vdash \exists x G(x)$ is an axiom of $\mathbb{ID}(\Pi)$, hence there exists a (trivial) proof of such a sequent in $\mathbb{ID}(\Pi)$. By constructivity, $\mathbb{ID}(\Pi)$ must also contain a proof τ of the sequent $\vdash \theta G(x)$, for some ground substitution θ. By the above considerations, it follows that the program $\mathcal{P}$ extracted from π is able to compute the answer substitution θ (or a more general one) for the goal $\leftarrow G(x)$.

To better formalize, we introduce the notion of evaluation.

Definition 8 (Evaluation). *Given a set of proofs Π (on $\mathcal{L}_\Sigma$) and a formula A of $\mathcal{L}_\Sigma$, A is* evaluated *in Π (in symbols $\Pi \rhd A$) if, for every ground substitution θ, one of the following inductive conditions holds:*

1. *θA is an atomic or a negated formula and $\vdash \theta A \in \mathrm{Seq}(\Pi)$;*
2. *$\theta A \equiv B \wedge C$ and $\Pi \rhd B$ and $\Pi \rhd C$;*
3. *$\theta A \equiv B \vee C$ and either $\Pi \rhd B$ or $\Pi \rhd C$;*
4. *$\theta A \equiv B \rightarrow C$ and, if $\Pi \rhd B$ then $\Pi \rhd C$;*
5. *$\theta A \equiv \forall x B(x)$ and, for every ground term t of $\mathcal{L}_\Sigma$, $\Pi \rhd B(t)$;*
6. *$\theta A \equiv \exists x B(x)$ and $\Pi \rhd B(t)$ for some ground term t of $\mathcal{L}_\Sigma$.*

A set Γ of formulas is evaluated in Π ($\Pi \rhd \Gamma$) if $\Pi \rhd A$ holds for every $A \in \Gamma$. The following fact can be proved:

Lemma 1. *Let Π be any recursive set of proofs of $\mathcal{ND}_\mathcal{C}(\mathcal{T})$ and let H be an extended program clause. If there exists a proof $\pi : \ \vdash H$ in the closure under substitution of $[\Pi]$, then $\mathbb{ID}(\Pi) \rhd H$.*

Proof. Let $H = A_1 \wedge \ldots \wedge A_n \rightarrow B$ be an extended program clause provable in the closure under substitution of $[\Pi]$, this implies that there exist a sequent $\vdash A'_1 \wedge \ldots \wedge A'_n \rightarrow B' \in \mathrm{Seq}([\Pi])$ and a substitution θ' such that $H \equiv \theta'(A'_1 \wedge \ldots \wedge A'_n \rightarrow B')$. Now, let us consider an arbitrary ground substitution θ; we must prove that, if $\mathbb{ID}(\Pi) \rhd \{\theta\theta' A'_1, \ldots, \theta\theta' A'_n\}$ then $\mathbb{ID}(\Pi) \rhd \theta\theta' B'$. But $\mathbb{ID}(\Pi) \rhd \theta\theta' A'_i$ for every $i = 1, \ldots, n$ implies, by definition of evaluation, that $\mathbb{ID}(\Pi)$ contains a proof $\tau_i : \ \vdash \theta\theta' A'_i$ for every $i = 1, \ldots, n$; moreover, $\mathbb{ID}(\Pi)$ contains also a proof

of $\vdash A'_1 \wedge \ldots \wedge A'_n \rightarrow B'$. We can apply the SLD$^+$-rule

$$\frac{\tau_1 : \ \vdash \theta\theta' A_1 \quad \ldots \quad \tau_n : \ \vdash \theta\theta' A_n \qquad \overline{\vdash A'_1 \wedge \ldots \wedge A'_n \rightarrow B'}}{\vdash \theta\theta' B'}\text{SLD}^+$$

to get a proof of $\theta\theta' B'$ in $\mathbb{ID}(\Pi)$. Since B' is atomic or negated, this implies that $\mathbb{ID}(\Pi) \rhd \theta\theta' B'$.

This immediately guarantees that every formula in the extended logic program $\mathcal{P}_\Pi$ extracted from a finite set of proofs $\Pi \subseteq \mathcal{ND}_\mathcal{C}(\mathcal{T})$ is evaluated in $\mathbb{ID}(\Pi)$.

Lemma 2. *Let Π be a recursive set of proofs of $\mathcal{ND}_\mathcal{C}(\mathcal{T})$ over the language $\mathcal{L}_\Sigma$ containing a finite set $\mathcal{C}$ of constant and function symbols, such that $\mathbb{ID}(\Pi) \rhd \mathcal{T}$. For every $\pi : \Gamma \vdash A$ belonging to the closure under substitution of $[\Pi]$, if $\mathbb{ID}(\Pi) \rhd \Gamma$ then $\mathbb{ID}(\Pi) \rhd A$.*

Proof. The proof goes by induction on the depth of π. Suppose $\text{depth}(\pi) = 0$; then either π is a proof of the sequent $A \vdash A$ (i.e., we introduce an assumption A) or π proves the sequent $\vdash A$ with $A \in \mathcal{T}$. In both cases the assertion immediately follows. Now, let us suppose that the assertion holds for any proof $\pi' : \Gamma' \vdash A'$ belonging to the closure under substitution of $[\Pi]$ such that $\text{depth}(\pi') \leq h$, and let us suppose that $\text{depth}(\pi) = h + 1$. The proof goes by cases according to the last rule applied in π. We only see some representative cases.

$\rightarrow$*-introduction.* In this case π has the following form:

$$\pi : \Gamma \vdash A \equiv \frac{\begin{array}{c}\Gamma, [B] \\ \vdots\ \pi_1 \\ C\end{array}}{B \rightarrow C}\rightarrow I$$

Let θ be any ground substitution. Since π_1 is a subproof of π, the proof $\theta\pi_1 \equiv \theta\Gamma, \theta B \vdash \theta C$ belongs to the closure under substitution of $[\Pi]$. Suppose that $\mathbb{ID}(\Pi) \rhd \theta B$; by applying the induction hypothesis on $\theta\pi_1$ (in fact, $\mathbb{ID}(\Pi) \rhd \theta\Gamma$), we get that $\mathbb{ID}(\Pi) \rhd \theta C$.

$\forall$*-introduction.* In this case π has the following form:

$$\pi : \Gamma \vdash A \equiv \frac{\begin{array}{c}\Gamma \\ \vdots\ \pi_p \\ B(p)\end{array}}{\forall x\ B(x)}\forall I$$

Let θ be any ground substitution. Since $\theta\pi_p : \theta\Gamma \vdash \theta B(p)$ belongs to the closure under substitution of $[\Pi]$, the proof $\theta\pi_p[t/p] : \theta\Gamma \vdash \theta B(t)$ belongs to the closure under substitution of $[\Pi]$ for every ground term t of $\mathcal{L}_\Sigma$ (we recall that under the usual assumptions on proper parameters of [15,17], p does not belong to the domain of θ and $\theta\pi_p$ is a correct proof). Hence, by induction hypothesis,

$\mathbb{ID}(\Pi) \rhd \theta B(t)$ for every ground term t of the language.

$\vee$-*elimination*. In this case π has the following form:

$$\pi : \Gamma \vdash A \equiv \dfrac{\begin{array}{ccc} \Gamma_1 & \Gamma_2, [B] & \Gamma_3, [C] \\ \vdots\, \pi_1 & \vdots\, \pi_2 & \vdots\, \pi_3 \\ B \vee C & A & A \end{array}}{A} \vee E$$

Let θ be any ground substitution. Since $\theta\pi_1$ belongs to the closure under substitution of $[\Pi]$, by induction hypothesis either $\mathbb{ID}(\Pi) \rhd \theta B$ or $\mathbb{ID}(\Pi) \rhd \theta C$. Therefore we can apply the induction hypothesis either to $\theta\pi_2$ or to $\theta\pi_3$ to deduce that $\mathbb{ID}(\Pi) \rhd \theta A$.

$\neg$-*introduction*. In this case π has the following form:

$$\pi : \Gamma \vdash A \equiv \dfrac{\begin{array}{c} [H_1, \ldots, H_n, K] \\ \vdots\, \pi_1 \\ B \wedge \neg B \end{array}}{H_1 \wedge \ldots \wedge H_n \rightarrow \neg K} I\neg$$

with $H_1, \ldots, H_n$ atomic or negated formulas. $H_1, \ldots, H_n, K$ are the only undischarged assumptions of π.

Since $H_1 \wedge \ldots \wedge H_n \rightarrow \neg K$ is an extended program clause, the assertion immediately follows from Lemma 1.

Cover Set Induction Rule. In this case $\pi : \Gamma \vdash A$ has the following form:

$$\pi : \Gamma \vdash A \equiv \dfrac{\begin{array}{ccc} \Gamma_1, [\Delta_1] & & \Gamma_n, [\Delta_n] \\ \vdots\, \pi_1 & & \vdots\, \pi_n \\ B(t_1) & \cdots & B(t_n) \end{array}}{B(x)} \mathrm{Ind}^{\mathcal{C}}$$

Let θ be any ground substitution; we have to prove that $\mathbb{ID}(\Pi) \rhd \theta B(t)$, for every ground term t of $\mathcal{L}_\Sigma$. We proceed by a secondary induction on the structure of t. Let $\mathcal{C}$ consist of the symbols $s_1, \ldots, s_n$; then, either t coincides with some constant symbol s_j or t has the form $s_j(t_1, \ldots, t_k)$, where s_j is a k-ary function symbol of $\mathcal{C}$ and $t_1, \ldots, t_k$ are ground terms. In the former case, there exists a subproof $\pi_j : \Gamma_j \vdash B(t_j)$ of π such that t_j coincides with s_j. Let us consider the proof $\theta\pi_j : \theta\Gamma_j \vdash \theta B(s_j)$; by applying the main induction hypothesis to $\theta\pi_j$ (which belongs to the closure under subproofs of $[\Pi]$), we get $\mathbb{ID}(\Pi) \rhd \theta B(s_j)$. Otherwise, let us take the subproof $\pi_j : \Gamma_j, B(x_1), \ldots, B(x_k) \vdash B(s_j(x_1, \ldots, x_k))$. The subproof $\pi'_j : \theta\Gamma_j, \theta B(t_1), \ldots, \theta B(t_k) \vdash \theta B(s_j(t_1, \ldots, t_k))$ belongs to the closure under substitution of $[\Pi]$ and, by the secondary induction hypothesis (since $t_1, \ldots, t_k$ are simpler than t), $\mathbb{ID}(\Pi) \rhd \theta B(t_1), \ldots, \mathbb{ID}(\Pi) \rhd \theta B(t_k)$. By the main induction hypothesis, we can conclude that $\mathbb{ID}(\Pi) \rhd \theta B(s_j(t_1, \ldots, t_k))$.

Now we give some conditions about correctness of programs.

Theorem 3. *Let $\langle D_r, \mathcal{G}_r \rangle$ be a specification in a closed framework $\mathcal{F}$ with a finite set of construction symbols $\mathcal{C}$, where D_r completely defines r; let P be*

an extended logic program over the language $\mathcal{L}_{\Sigma\cup\{r\}}$ *having* $\mathcal{C}$ *as constant and function symbols, such that:*

1. *There exists a proof* π *of the formula* $\forall\underline{x}(\exists\underline{y}\ r(\underline{x},\underline{y}) \vee \neg\exists\underline{y}\ r(\underline{x},\underline{y}))$ *in the calculus* $\mathcal{ND}_{\mathcal{C}}(P)$*;*
2. P *is valid in the canonical model* $\mathfrak{I}_r$ *of* $\mathcal{F}\cup\{D_r\}$*.*

Then, the extended logic program $\mathcal{P}^*$ *extracted from* π *is totally correct in* $\mathcal{F}\cup\{D_r\}$*.*

Proof. Let $\underline{t}$ be a tuple of ground terms of $\mathcal{L}_{\Sigma\cup\{r\}}$. Let us suppose that σ is a computed answer substitution for $\mathcal{P}^*\cup\{\leftarrow r(\underline{t},\underline{y})\}$. We have $\mathcal{P}^*\models\sigma r(\underline{t},\underline{y})$; moreover, by the fact that $\mathfrak{I}_r$ is a model of P, we also have that $\mathfrak{I}_r$ is a model of $\mathcal{P}^*$. We can conclude $\mathfrak{I}_r\models\sigma r(\underline{t},\underline{y})$, and this proves the correctness of $\mathcal{P}^*$. To prove the completeness, let us suppose that $\exists\underline{y}\ r(\underline{t},\underline{y})$ is valid in the canonical model $\mathfrak{I}_r$. We have to show that, for some tuple $\underline{s}$ of ground terms of $\mathcal{L}_{\Sigma\cup\{r\}}$, $\mathcal{P}^*\cup\{\leftarrow r(\underline{t},\underline{s})\}$ has a SLD$^+$-refutation. This amounts to showing that $r(\underline{t},\underline{s})$ is provable in $\mathbb{ID}(\Pi)$, where $\Pi=\{\pi\}$. We know, by Lemma 1, that $\mathbb{ID}(\Pi)\rhd P$; we can apply Lemma 2 and state that $\mathbb{ID}(\Pi)\rhd\forall\underline{x}(\exists\underline{y}\ r(\underline{x},\underline{y})\vee\neg\exists\underline{y}\ r(\underline{x},\underline{y}))$, which implies $\mathbb{ID}(\Pi)\rhd\exists\underline{y}\ r(\underline{t},\underline{y})\vee\neg\exists\underline{y}\ r(\underline{t},\underline{y})$, thus either $\mathbb{ID}(\Pi)\rhd\exists\underline{y}\ r(\underline{t},\underline{y})$ or $\mathbb{ID}(\Pi)\rhd\neg\exists\underline{y}\ r(\underline{t},\underline{y})$. It follows that either $\mathbb{ID}(\Pi)\rhd r(\underline{t},\underline{s})$ for some tuple $\underline{s}$ of ground terms of $\mathcal{L}_{\Sigma\cup\{r\}}$ or $\mathbb{ID}(\Pi)\rhd\neg\exists\underline{y}\ r(\underline{t},\underline{y})$. On the other hand, it is not the case that $\mathbb{ID}(\Pi)\rhd\neg\exists\underline{y}\ r(\underline{t},\underline{y})$; otherwise, it would follow that $\neg\exists\underline{y}\ r(\underline{t},\underline{y})$ is valid in $\mathfrak{I}_r$, a contradiction. Thus, we have proven that $\mathbb{ID}(\Pi)\rhd r(\underline{t},\underline{s})$ for some $\underline{s}$. By definition of evaluation, $\mathbb{ID}(\Pi)$ contains a proof τ of $r(\underline{t},\underline{s})$.

The above proof relies on the fact that formulas are evaluated in a constructive sense in $\mathbb{ID}(\Pi)$. To get this the choice of the natural calculus $\mathcal{ND}_{\mathcal{C}}(P)$ is essential; for instance, if we take the classical calculus, then a proof π of $\forall\underline{x}(\exists\underline{y}\ r(\underline{x},\underline{y})\vee\neg\exists\underline{y}\ r(\underline{x},\underline{y}))$ always exists, but in general we have no means to evaluate it in a constructive sense in $\mathbb{ID}(\Pi)$. We also remark that there is a close relation between the program $\mathcal{P}^*$ extracted from the proof π and the program P; indeed, $\mathcal{P}^*$ contains the axioms of P which are relevant for the computations of the goals in $\mathcal{G}_r$, moreover it may contain other relevant clauses occurring in the proof π.

If the relation r is total (i.e., for every tuple $\underline{t}$ there is a tuple $\underline{s}$ such that $r(\underline{t},\underline{s})$ holds), it is preferable to provide a proof of $\forall\underline{x}\exists\underline{y}\ r(\underline{x},\underline{y})$ in order to apply the previous theorem since this makes easier the reusability of the extracted program.

6 Some Examples

Let us see some examples in the framework $\mathcal{PA}$, having, as canonical model, the standard model $\mathfrak{N}$ of Arithmetic. Suppose we have to solve a decision problem, for instance, to decide whether a number is even. We introduce a new unary relation symbol p together with the definition D_p

$$p(x)\leftrightarrow\exists y(x=y+y)$$

Clearly D_p completely defines p in $\mathcal{PA}$. The set of goals $\mathcal{G}_p$ contains the atoms of the form $p(0), p(s(0)), p(s(s(0)), \ldots$ (we recall that the constructions of $\mathcal{PA}$ are $0, s(0), \ldots$). To synthesize a program P_{even} which is totally correct in $\mathcal{PA} \cup \{D_p\}$, we have to find an extended logic program P such that P is valid in $\mathfrak{N}$ and a proof

$$\pi : \ \vdash \forall x(p(x) \vee \neg p(x))$$

exists in the natural calculus $\mathcal{ND}_{\mathcal{C}}(P)$, where $\mathcal{C} = \{0, s\}$.

We try to get the proof π by applying the Cover Set Induction Rule induced by $\mathcal{C}$; to this end, we need a proof π_1 of $p(0) \vee \neg p(0)$ and a proof π_2 of $p(s(x)) \vee \neg p(s(x))$ which depends on $p(x) \vee \neg p(x)$. Note that, since $\mathcal{C}$ is a set of constructions for $\mathcal{PA}$, the induction schema associated with $\mathcal{C}$ is valid in the model $\mathfrak{N}$, the intended semantics of $\mathcal{PA}$. Thus, we use the Cover Set Induction Rule as an heuristic to single out the axioms we need to build π. Of course, one could also strengthen the natural calculus with new rules; on the other hand one has to revise the previous treatment to guarantee the validity of Theorem 3.

To build the proof π_1, we observe that $p(0)$ is valid in $\mathfrak{N}$. Let us take

$$(E_1)\ p(0) \qquad P := \{(E_1)\}$$

By applying the rule $\vee I_l$ to (E_1), we get a proof of $p(0) \vee \neg p(0)$ in $\mathcal{ND}_{\mathcal{C}}(P)$. Now we attempt to build π_2 according to the following schema:

$$\dfrac{\begin{array}{cccc} & & [p(x)] & [\neg p(x)] \\ & & \vdots\ \pi_3 & \vdots\ \pi_4 \\ p(x) \vee \neg p(x) & p(s(x)) \vee \neg p(s(x)) & p(s(x)) \vee \neg p(s(x)) & p(s(x)) \vee \neg p(s(x)) \end{array}}{p(s(x)) \vee \neg p(s(x))}\ \vee E$$

To complete π_3, we introduce the following axiom valid in $\mathfrak{N}$

$$(E_2)\ p(x) \rightarrow \neg p(s(x))$$

which allows us to derive, by applying $\rightarrow E$, $\neg p(s(x))$ and then, by $\vee I_r$, $p(s(x)) \vee \neg p(s(x))$. Thus, adding (E_2) to P, the proof π_3 belongs to $\mathcal{ND}_{\mathcal{C}}(\mathcal{T})$. The proof π_4 is symmetric and we have to take

$$(E_3)\ \neg p(x) \rightarrow p(s(x)) \qquad P := P \cup \{(E_3)\}$$

At this point, the proof π_1 is completely defined; in virtue of Theorem 3, the general logic program P_{even} extracted from π is totally correct for $\mathcal{PA} \cup \{D_p\}$ and solves our decision problem; one can easily check that:

$$P_{even} = \{(E_1), (E_2), (E_3)\}$$

Suppose now we have to compute a function, for instance, the function $f(x) = x+x$. In this case, we have to introduce a binary relation r with the definition D_r:

$$r(x, y) \leftrightarrow y = x + x$$

where D_r completely specifies r and the set $\mathcal{G}_r$ is

$$\mathcal{G}_r = \{\exists y\ r(0,y), \exists y\ r(s(0),y), \exists y\ r(s(s(0)),y), \ldots\}$$

In this case, since the function f is total, we provide a proof

$$\pi : \ \vdash \forall x \exists y\ r(x,y)$$

We build π according to the schema

$$\dfrac{\begin{array}{cc} & [\exists y\ r(x,y)] \\ \vdots\ \pi_2 & \vdots\ \pi_3 \\ \exists y\ (r(0,y)) & \exists y\ r(s(x),y) \end{array}}{\exists y\ r(x,y)}\ \mathrm{Ind}^{\mathcal{C}}$$

Let us set

$$(F_1)\ r(0,0) \qquad P := \{(F_1)\}$$

It is easy to build π_2 in $\mathcal{ND}_{\mathcal{C}}(P)$. As regards π_3, we observe that the formula

$$(F_2)\ r(x,y) \rightarrow r(s(x), s(s(y)))$$

is valid in $\mathfrak{N}$. We can use (F_2) to derive $\exists y\ r(s(x),y)$ from the assumptions $\exists y\ r(x,y)$; thus, adding (F_2) to P, the proof π_3 belongs to $\mathcal{ND}_{\mathcal{C}}(P)$. Since the proof π has been completed, our synthesis process can be halted and the program extracted from π coincides with P.

We could solve our synthesis process in another way. Let us enrich our specification by a new ternary relation *sum* with definition D_{sum}

$$sum(x_1, x_2, y) \leftrightarrow y = x_1 + x_2$$

and a goal set

$$\mathcal{G}_{sum} = \{\exists y\ sum(0,0,y), \exists y\ sum(s(0),0,y), \exists y\ sum(0,s(0),y), \ldots\}$$

To find a program for $\mathcal{G}_{sum}$, since *sum* is a total function, a proof

$$\pi^* : \ \vdash \forall x_1 \forall x_2 \exists y\ sum(x_1, x_2, y)$$

is needed. Suppose that π^* has already been given in some calculus $\mathcal{ND}_{\mathcal{C}}(S)$ so to satisfy the hypothesis of Theorem 3; then a program for $\mathcal{G}_r$ can be easily synthesized. As a matter of fact, let us take

$$(F^*)\ sum(x,x,y) \rightarrow r(x,y) \qquad P := S \cup \{(F^*)\}$$

We can build the following proof in $\mathcal{ND}_{\mathcal{C}}(P)$:

$$\dfrac{\dfrac{\dfrac{\vdots\ \pi^*}{\forall x_1 \forall x_2 \exists y\ sum(x_1,x_2,y)}\ \forall E}{\exists y\ sum(x,x,y)} \qquad \dfrac{\dfrac{[sum(x,x,z)] \quad \dfrac{}{sum(x,x,z) \rightarrow r(x,z)}\ I_P}{r(x,z)}\ \rightarrow E}{\exists y\ r(x,y)}\ \exists I}{\exists y\ r(x,y)}\ \exists E$$
$$\dfrac{\exists y\ r(x,y)}{\forall x \exists y\ r(x,y)}\ \forall I$$

Since P is valid in $\mathfrak{N}$, by Theorem 3 the program $\mathcal{P}$ extracted from π is totally correct in $\mathcal{PA} \cup D_r \cup D_{sum}$. Note that $\mathcal{P}$ consists of the union of the program S^* extracted from π^* and the axiom (F^*), which is the only extended program clause extracted from the right-hand subproof. Thus, provided that we are able to compute sum, we can compute $r(x, y)$. Now, we can treat apart the task of finding a program for sum. We can proceed as above and we can show that the extended program

$$S = \left\{ \begin{array}{l} sum(0, 0, 0) \\ sum(x, y, z) \rightarrow sum(s(x), y, s(z)) \end{array} \right\}$$

suffices to build the proof π^* and the extracted program S^* coincides with S. This example shows how one can combine programs computing different functions and reuse them. We can go on with this process considering also the cases where the definition axiom D_r is a complex formula. In this case, to find the appropriate program, it might be necessary many steps of introductions of new relations. This leads to a sequence of of "specifications" $\mathcal{D}_0 \subseteq \mathcal{D}_1 \subseteq \ldots$ and of extended logic programs $\mathcal{P}_0 \subseteq \mathcal{P}_1 \subseteq \ldots$, where with each $\mathcal{D}_k$ is associated a set of goals $\mathcal{G}_k$. When conditions of Theorem 3 can be applied, this synthesis process can be halted and this provides a halting criterion (see also [10]).

References

1. M. Benini. *Verification and Analysis of Programs in a Constructive Environment*. PhD thesis, Dipartimento di Scienze dell'Informazione, Università di Milano, 1999. 4
2. S. Buss and G. Mints. The complexity of the disjunction and existential properties in intuitionistic logic. *Annals of Pure and Applied Logic*, 99(3):93–104, 1999. 4
3. Y. Deville and K. Lau. Logic program synthesis. *Journal of Logic Programming*, 19(20):321–350, 1994. 1, 9
4. M. Ferrari. *Strongly Constructive Formal Systems*. PhD thesis, Dipartimento di Scienze dell'Informazione, Universitá degli Studi di Milano, Italy, 1997. Available at `http://homes.dsi.unimi.it/~ferram`. 1, 4
5. M. Ferrari, C. Fiorentini, and P. Miglioli. Goal oriented information extraction in uniformly constructive calculi. In *Proceedings of WAIT'99: Workshop Argentino de Informática Teórica*, pages 51–63. Sociedad Argentina de Informática e Investigación Operativa, 1999. 1, 2, 3, 4
6. M. Ferrari, C. Fiorentini, and P. Miglioli. Extracting information from intermediate T-systems. Technical Report 252-00, Dipartimento di Scienze dell'Informazione, Universitá degli Studi di Milano, Italy, 2000. Available at `http://homes.dsi.unimi.it/~ferram`. 1, 2, 4
7. M. Ferrari, C. Fiorentini, and P. Miglioli. Extracting information from intermediate semiconstructive HA-systems (extended abstract). *Mathematical Structures in Computer Science*, 11, 2001. 1, 2, 4
8. M. Ferrari, P. Miglioli, and M. Ornaghi. On uniformly constructive and semiconstructive formal systems. Submitted to *Annals of Pure and Applied Logic*, 1999. 1, 2, 4

9. K.-K. Lau and M. Ornaghi. On specification frameworks and deductive synthesis of logic programs. In Logic Program Synthesis and Transformation. *Proceedings of LOPSTR'94*. Springer-Verlag, 1994. 1, 2, 5
10. Kung-Kiu Lau, Mario Ornaghi, and Sten-Åke Tärnlund. The halting problem for deductive synthesis of logic programs. In P. Van Hentenryck, editor, *Logic Programming - Proceedings of the Eleventh International Conference on Logic Programming*, pages 665–683. The MIT Press, 1994. 1, 2, 5, 16
11. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 2nd edition, 1987. 5, 7
12. P. Miglioli, U. Moscato, and M. Ornaghi. Abstract parametric classes and abstract data types defined by classical and constructive logical methods. *Journal of Symbolic Computation*, 18:41–81, 1994. 1
13. 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. 4
14. A. Momigliano and M. Ornaghi. Regular search spaces as a foundation of logic programming. In R. Dyckhoff, editor, *Proceedings of the 4th International Workshop on Extensions of Logic Programming*, volume 798 of *LNAI*, pages 222–254, Berlin, 1994. Springer. 9
15. D. Prawitz. *Natural Deduction*. Almquist and Winksell, 1965. 7, 9, 11
16. J. C. Shepherdson. Negation in logic progamming. In J. Minker, editor, *Found. of Deductive Databases and Logic Programming*, page 19. Morgan Kaufmann, San Mateo, CA, 1988. 6
17. A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996. 9, 11