

# A Logic Programming Approach to the Verification of Functional-Logic Programs \*

José Miguel Cleva  
jcleva@sip.ucm.es

Javier Leach  
leach@sip.ucm.es

Francisco J.  
López-Fraguas  
fraguas@sip.ucm.es

Departamento de Sistemas Informáticos y Programación  
Facultad de Informática  
Univerisdad Complutense de Madrid  
28040 Madrid, Spain

## ABSTRACT

We address in this paper the question of how to verify program properties in modern functional logic languages, where it is allowed the presence of non-deterministic functions with call-time choice semantics. The main problem to face is that for such kind of programs equational reasoning is not valid. We develop some logical conceptual tools providing sound reasoning mechanisms for these programs, in particular for proving properties valid in the initial model of a program. We show how CRWL, a well known logical framework for functional logic programming, can be easily mapped into logic programming, and we use this mapping as a starting point of our work. We explore then how to prove properties of the resulting logic programming translation by means of different existing interactive proof assistants, and afterwards we give some proposals trying to overcome the limitations of the approach, specially with respect to its theoretical strength.

## Categories and Subject Descriptors

D.1.1 [Programming Techniques]: Applicative (Functional) Programming; D.1.6 [Programming Techniques]: Logic Programming; D.2.4 [Software Engineering]: Software/Program Verification—*Formal methods*; F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs—*Logics of programs*

## General Terms

Theory, Verification, Languages

\*The authors have been partially supported by the Spanish CICYT (project TIC 2002-01167 ‘MELODIAS’)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PPDP’04, August 24–26, 2004, Verona, Italy.

Copyright 2004 ACM 1-58113-819-9/04/0008 ...\$5.00.

## Keywords

Functional logic programming, Verification, Logic programming

## 1. INTRODUCTION

One distinguished feature of modern functional logic languages like Curry [17] or Toy [19] is that programs are constructor based rewrite systems allowed to be non-terminating and non-confluent. Semantically this leads to the presence of non-strict and non-deterministic functions, which have been shown quite useful for practical declarative programming.

However, equational reasoning is non valid for reasoning about programs because of the non-determinism. The *CRWL* (Constructor-based ReWriting Logic) framework [12, 13] - which is the theoretical basis of our work - gives a well-established alternative logic for functional logic programming (FLP). In *CRWL* the semantics of a program is given by its possible reductions, expressed by means of a reducibility relation  $e \rightarrow t$  between evaluable expressions and constructor terms, which are the sensible kind of results of computations. *CRWL* provides a proof calculus prescribing which reduction statements  $e \rightarrow t$  hold for a given program.<sup>1</sup> Programs have initial models, which are commonly accepted as the natural candidates to be intended models of programs.

*CRWL* has been successfully extended to cope with many other features relevant to productive programming: HO, objects, subsorts, algebraic datatypes, constraints and failure. See [28] for a recent survey of the *CRWL*-approach to FLP. Here we restrict ourselves to first order programs.

Verification of properties of logic and functional programs has been frequently studied [27, 26, 16]. We do not know many results in the FLP setting. The work of Padawitz [23, 24] in equational logic programming constitute a serious effort, both at the theoretical and the practical level. In Padawitz functions are deterministic and with strict semantics. There is some other work contemplating the issue of FLP program properties from a specific point of view. This includes different topics about declarative debugging [7, 8, 1], abstract interpretation [6] or abstract diagnosis [2].

<sup>1</sup>*CRWL* considers also a different kind of semantic statements, called joinability statements, which are useful for a good treatment of strict equality, a matter which we do not consider here.

The goal of our paper is to develop a logical basis from which quite general properties of FLP programs (like those of Curry or Toy) can be formulated and proved. The main lines of our approach can be summarized in advance as follows:

- Programs are *CRWL*-programs and the properties of interest are those valid in the initial model of a given program  $P$ , expressed as first order logic (FOL) formulas with reduction ( $\rightarrow$ ) as relation symbol.
- The *CRWL*-semantics of  $P$  is expressed by means of a FOL theory, which actually is a logic program  $P_L$ , whose least model corresponds closely to the *CRWL*-initial model of  $P$ .
- We can prove properties valid in those models by FOL deduction from a FOL theory consisting of the completion of  $P_L$  extended with inductive axioms. The set of provable valid properties can be enhanced by refining this theory, in particular by embedding in it some meta-theory about *CRWL*-derivations.

The remainder of the paper is organized as follows. The next section presents some preliminaries about *CRWL*. In section 3 we draw a parallel FOL theory  $P_L$  – a logic program indeed – for any given *CRWL*-program  $P$  such that *CRWL*-deducibility from  $P$  corresponds to FOL-logical consequence from  $P_L$ . In section 4, in order to prove properties of the initial model of a sample *CRWL*-program  $P$ , we translate the inductive extension of the completion of  $P_L$  into several existing interactive proof assistants. In section 5 we introduce a variant of the logic program  $P_L$  emulating *CRWL*, where the derivation trees for statements  $e \rightarrow t$  are explicit. Finally, section 6 summarizes some conclusions.

## 2. PRELIMINARIES: CRWL PROGRAMS AND THEIR LOGICAL SEMANTICS

We assume a signature  $\Sigma = CS_\Sigma \cup FS_\Sigma$  where  $CS_\Sigma = \bigcup_{n \in \mathbb{N}} CS_\Sigma^n$  is a set of *constructor* symbols and  $FS_\Sigma = \bigcup_{n \in \mathbb{N}} FS_\Sigma^n$  is a set of *function* symbols, all of them with associated arity and such that  $CS_\Sigma \cap FS_\Sigma = \emptyset$ . We also assume a countable set  $\mathcal{V}$  of *variable* symbols. We write  $Exp_\Sigma$  for the set of (total) *expressions* built up with  $\Sigma$  and  $\mathcal{V}$  in the usual way, and we distinguish the subset  $CTerm_\Sigma$  of (total) constructor terms or (total) *c-terms*, which only make use of  $CS_\Sigma$  and  $\mathcal{V}$ . The subindex  $\Sigma$  will usually be omitted. Expressions intend to represent possibly reducible expressions, while *c-terms* represent not further reducible data values.

The signature  $\Sigma_\perp$  results of extending  $\Sigma$  with the new constant (0-arity constructor)  $\perp$ , that plays the role of the undefined value. The sets  $Exp_\perp$  and  $CTerm_\perp$  of (partial) expressions and (partial) *c-terms* respectively are built up using  $\Sigma_\perp$ . Partial *c-terms* represent the result of partially evaluated expressions; thus, they can be seen as approximations (in a suitable information ordering  $\sqsubseteq$  defined bellow) to the value of expressions. A partial *c-term* is called *ground* if it does not contain any variable.

As usual notation we will write  $X, Y, Z, \dots$  for variables,  $c, d$  for constructor symbols,  $f, g$  for functions,  $e$  for expressions and  $s, t$  for *c-terms*. In all cases, primes (') and subindices can be used. Expressions can be compared by the *approximation ordering*  $\sqsubseteq$ , defined as the least partial

ordering verifying:  $\perp \sqsubseteq e$  and  $e_1 \sqsubseteq e'_1 \wedge \dots \wedge e_n \sqsubseteq e'_n \Rightarrow h(e_1, \dots, e_n) \sqsubseteq h(e'_1, \dots, e'_n)$ , for  $h \in CS^n \cup FS^n$ .

We will use the sets of substitutions  $CSubst = \{\theta : \mathcal{V} \rightarrow CTerm\}$  and  $CSubst_\perp = \{\theta : \mathcal{V} \rightarrow CTerm_\perp\}$ . We write  $e\theta$  for the result of applying  $\theta$  to  $e$ .

In the next sections we will need some familiar notions about first order logic and logic programming (see e.g. [11, 5] for standard references). We will use  $\varphi, \varphi', \dots$  for FOL-formulas and the standard notation  $T \models \varphi, I \models \varphi$  for logical consequence from a FOL theory (i.e., set of formulas)  $T$  and validity in a given interpretation  $I$ . We write also  $I \models T$  to indicate that  $I$  is a model of  $T$ .

### 2.1 The Proof Calculus for CRWL

In this work a *CRWL-program*  $\mathcal{P}$  is a finite set of rewrite rules of the form  $f(t_1, \dots, t_n) \rightarrow e$  where  $f \in FS^n$ ,  $(t_1, \dots, t_n)$  is a linear tuple (each variable in it occurs only once) of *c-terms*, and  $e$  is an expression.<sup>2</sup> Notice that  $e$  can contain variables not occurring in  $f(t_1, \dots, t_n)$ . We write  $\mathcal{P}_f$  for the set of defining rules of  $f$  in  $\mathcal{P}$ .

From a given program  $\mathcal{P}$ , the proof calculus for *CRWL* can derive *reduction or approximation statements* of the form  $e \rightarrow t$ , with  $e \in Exp_\perp$  and  $t \in CTerm_\perp$ . The intended meaning of such statement is that  $e$  can be reduced to  $t$ , where reduction may be done by applying rewriting rules of  $\mathcal{P}$  or by replacing subterms of  $e$  by  $\perp$ . If  $e \rightarrow t$  can be derived,  $t$  represents one of the possible values of the denotation of  $e$ .

When using a function rule  $R$  to derive statements, the calculus uses the so called *c-instances* of  $R$ , defined as  $[R]_\perp = \{R\theta \mid \theta \in CSubst_\perp\}$ . We write  $[P]_\perp$  for the set of *c-instances* of all the rules of a program  $P$ . Parameter passing in function calls are expressed by means of these *c-instances* in the proof calculus.

Table 1 shows the proof calculus for *CRWL*. We write  $\mathcal{P} \vdash_{CRWL} \varphi$  for expressing that the statement  $\varphi$  is provable from the program  $\mathcal{P}$  with respect to this calculus. The rule (FR) allows to use *c-instances* of program rules to prove approximations. These *c-instances* may contain  $\perp$  and by rule (BT) any expression can be reduced to  $\perp$ . This reflects a non-strict semantics, allowing non-terminating programs to be meaningful. The condition  $t \neq \perp$  is imposed in (FR) just to avoid unnecessary derivations, because the case  $t = \perp$  is already covered in the rule (BT).

A distinguished feature of *CRWL* (shared by concrete systems like Curry or Toy) is that programs can be non-confluent, defining thus *non-deterministic functions*. As a typical example, consider the program (called *Coin* for future references) in Fig.1, which assumes the constructors 0 and  $s$  for natural numbers.

Notice that *coin* is a non-deterministic function, for which the previous calculus can derive the statements  $coin \rightarrow 0$  and  $coin \rightarrow s(0)$ . The use of *c-instances* in rule (FR) instead of general instances corresponds to *call time choice* semantics for non-determinism [18, 12, 13]). In the example, it is possible to build a *CRWL*-proof for  $double(coin) \rightarrow 0$  and

<sup>2</sup>In [12, 13] programs are made of conditional rules, where conditions are conjunctions of joinability (or strict equality) conditions. Since we are not dealing here with strict equality as a specific, built-in construct, and it is known [4, 29] that in programs like ours (following constructor discipline) conditions can be replaced by semantically equivalent *if-then* expressions, we consider here programs with non-conditional rules.

Table 1: Rules for  $CRWL$ -provability

(BT) Bottom	$\frac{}{e \rightarrow \perp}$	
(DC) Decomposition	$\frac{e_1 \rightarrow t_1, \dots, e_n \rightarrow t_n}{c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n)}$	$c \in CS^n, \quad t_i \in CTerm_{\perp}$
(FR) Function reduction	$\frac{e_1 \rightarrow t_1, \dots, e_n \rightarrow t_n \quad e \rightarrow t}{f(e_1, \dots, e_n) \rightarrow t}$	if $t \not\equiv \perp, f(t_1, \dots, t_n) \rightarrow e \in [P]_{\perp}$

$0 + Y \rightarrow Y$	$coin \rightarrow 0$
$s(X) + Y \rightarrow s(X + Y)$	$coin \rightarrow s(0)$
$double(X) \rightarrow X + X$	

Figure 1:  $CRWL$  sample program *Coin*

also for  $double(coin) \rightarrow s(s(0))$ , but not for  $double(coin) \rightarrow s(0)$ . This semantic option is not a caprice of  $CRWL$ . Call-time choice is related to *sharing*, a well known operational technique considered essential for the effective implementation of lazy functional languages like Haskell. Existing FLP languages like Curry or Toy also use sharing and call-time choice semantics. The above described behaviour for the reduction of  $double(coin)$  corresponds exactly with what happens in those systems. Run-time choice, an alternative semantics for non-determinism with which  $double(coin)$  can be reduced also to  $s(0)$  is investigated for the FLP setting in [3].

From the point of view of verifying properties of FLP programs, non-determinism and call-time choice semantics have the unpleasant consequence that equational reasoning is not valid for  $CRWL$ -programs. In the previous example, if the rules for *coin* were understood as the equalities  $coin = 0$  and  $coin = s(0)$ , then we could deduce  $0 = s(0)$ , which is not intended. Call-time choice implies that not only equational reasoning, but also ordinary rewriting is invalid since, from the point of view of rewriting, the rule  $double(X) \rightarrow X + X$  should be applicable to *any*  $X$ , and not only to c-terms. Hence, we would have  $double(coin) \rightarrow coin + coin$ , and from this,  $double(coin) \rightarrow s(0)$ , which is not valid with call-time choice.

A remark about the  $CRWL$ -calculus presented here, with respect to the original in [12, 13]: in addition to the above mentioned elimination of joinability statements, we have also dropped for technical reasons the so called *restricted reflexivity rule*:

$$(RR) \frac{}{X \rightarrow X} \quad X \in \mathcal{V}$$

At the end of this section we argue the advantages of having done so. But we first discuss the relation between both calculi. Inside this discussion, let us call  $CRWL$  the calculus of table 1, and  $CRWL_{RR}$  the proof calculus with the rule (RR). Within  $CRWL_{RR}$  we can prove, for instance,  $0 + X \rightarrow X$  and all its c-instances while in  $CRWL$  only the ground c-instances  $0 + t \rightarrow t$ , for any ground partial c-term  $t$ . The next result makes precise the relation between both calculi:

PROPOSITION 1. *Let  $P$  be a  $CRWL$ -program. Then:*

- (i)  $P \vdash_{CRWL} e \rightarrow t \Rightarrow P \vdash_{CRWL_{RR}} e \rightarrow t$
- (ii)  $P \vdash_{CRWL_{RR}} e \rightarrow t \Rightarrow P \vdash_{CRWL} e' \rightarrow t'$ , for all ground c-instances  $e' \rightarrow t'$  of  $e \rightarrow t$

PROOF. (i): Trivial, since  $CRWL$  is a part of  $CRWL_{RR}$ .

(ii): We reason by structural induction on the rules of the  $CRWL_{RR}$  derivation:

**Rule BT:**  $P \vdash_{CRWL_{RR}} e \rightarrow \perp$ , and then as **BT** is also a  $CRWL$  rule,  $P \vdash_{CRWL} e' \rightarrow \perp$  for every ground c-instance  $e'$  of  $e$ .

**Rule RR:** Let  $P \rightarrow_{CRWL_{RR}} X \rightarrow X$ , then the ground c-instances of  $X$  are the ground c-terms. We must prove  $P \vdash_{CRWL} t \rightarrow t$  for every ground c-term  $t$ , what is easy by induction on the structure of  $t$ .

**Rule DC:** Let  $e = c(e_1, \dots, e_n)$  and  $t = c(t_1, \dots, t_n)$  and  $P \vdash_{CRWL_{RR}} e_i \rightarrow t_i$  for every  $i$ . By the induction hypothesis,  $P \vdash_{CRWL} e'_i \rightarrow t'_i$  for every  $i$ . Every ground c-instance of  $e$  is formed by the ground c-instances of every  $e_i$ . Therefore, using the **DC** rule, we have  $P \vdash e' \rightarrow t'$  for every ground c-instances  $e' \rightarrow t'$  of  $e \rightarrow t$ .

**Rule FR:** We use the following known results for  $CRWL_{RR}$ :

LEMMA 1.  $P \vdash_{CRWL_{RR}} e \rightarrow t \Rightarrow P \vdash_{CRWL_{RR}} e\sigma \rightarrow t\sigma$  for every  $\sigma \in CSubst_{\perp}$

LEMMA 2. *If  $P \vdash_{CRWL_{RR}} e \rightarrow t$  and  $var(e) = var(t) = \emptyset$  then we have a derivation free of variables for  $e \rightarrow t$*

Suppose  $P \vdash_{CRWL_{RR}} f(e_1, \dots, e_n) \rightarrow t$ . Every ground c-instance  $f(e'_1, \dots, e'_n) \rightarrow t'$  of  $f(e_1, \dots, e_n) \rightarrow t$  comes from a substitution  $\sigma \in CSubst_{\perp}$  applied to  $f(e_1, \dots, e_n) \rightarrow t$ . Then we have  $P \vdash_{CRWL_{RR}} f(e'_1, \dots, e'_n) \rightarrow t'$  by lemma 1. Both  $f(e'_1, \dots, e'_n)$  and  $t'$  are ground and then, by lemma 2, we have a derivation for  $f(e'_1, \dots, e'_n) \rightarrow t'$  without variables. As the **RR** rule is only applied to variables, the obtained derivation is a  $CRWL$  proof and therefore  $P \vdash_{CRWL} f(e'_1, \dots, e'_n) \rightarrow t'$ .  $\square$

With respect to models the situation is the following. In  $CRWL_{RR}$  Herbrand models of programs have as support a Herbrand universe of partial c-terms with variables [12, 13], and every program  $P$  has a least Herbrand model  $M_{RR,P}$  which technically is a *free* model, while with  $CRWL$  as has

been presented here we must use the ordinary Herbrand universe of ground c-terms, and it can be shown that every program  $P$  has a least Herbrand model  $M_P$  which is an initial model. Least models verify:

PROPOSITION 2. For any CRWL-program  $P$ ,

- (i)  $P \vdash_{CRWL_{RR}} e \rightarrow t \Leftrightarrow M_{RRP} \models e \rightarrow t$
- (ii)  $P \vdash_{CRWL} e \rightarrow t \Leftrightarrow M_P \models e \rightarrow t$ , for any ground  $e \rightarrow t$
- (iii)  $M_{RRP} \models e \rightarrow t \Rightarrow M_P \models \forall(e \rightarrow t)$ , where  $\forall\varphi$  indicates the universal closure of  $\varphi$

PROOF SKETCH. Part (i) is proved in [13], (ii) is proved similarly, and (iii) is a consequence of both.  $\square$

We believe that, in some sense,  $M_P$  is more natural than  $M_{RRP}$  as intended model whose properties are to be formally verified. For instance, in the *Coin* example above, the property  $\varphi \equiv \forall E, T. (E \rightarrow T \Rightarrow E + 0 \rightarrow T)$ , which intuitively is a true property about addition and reduction, is in fact valid in  $M_P$ , but not in  $M_{RRP}$ , because with *RR* we can CRWL-prove  $X \rightarrow X$  (and then  $M_{RRP} \models X \rightarrow X$ ), but not  $X + 0 \rightarrow X$  (and then  $M_{RRP} \not\models X + 0 \rightarrow X$ ).

### 3. CRWL AS A LOGIC PROGRAM

In this section we will map CRWL into first order logic (FOL). We assume the reader is familiar with standard notions of FOL (see e.g.[11]) and logic programming (see e.g. [5]). We want to associate to a given CRWL-program  $P$  a FOL theory  $P_L$  such that CRWL-deducibility from  $P$  corresponds to FOL-logical consequence from  $P_L$ . The theory  $P_L$  will be indeed a logic program, and we will use this logic program to prove properties of the original CRWL program as stated by the results given in this section.

Consider a CRWL program  $P$  with signature  $\Sigma = CS \cup FS$ . The logic program  $P_L$  associated with  $P$  is made of the following clauses (written as implications  $l \Leftarrow C_1 \wedge \dots \wedge C_n$ ,  $n \geq 0$ ) defining the relation  $\rightarrow$ :

$$\perp \rightarrow \perp$$

For every  $c \in CS$ :

$$\begin{aligned} c(E_1, \dots, E_n) \rightarrow \perp \\ c(E_1, \dots, E_n) \rightarrow c(T_1, \dots, T_n) \Leftarrow \\ E_1 \rightarrow T_1 \wedge \dots \wedge E_n \rightarrow T_n \end{aligned}$$

For every  $f \in FS$ :

$$f(E_1, \dots, E_n) \rightarrow \perp$$

For every rule  $f(t_1, \dots, t_n) = e \in P$ :

$$f(E_1, \dots, E_n) \rightarrow T \Leftarrow \\ E_1 \rightarrow t_1 \wedge \dots \wedge E_n \rightarrow t_n \wedge e \rightarrow T$$

Since  $P_L$  is a logic program, we may consider for it standard notions, like that of *completion* of  $P_L$ ,  $Comp(P_L)$ . The following are well known results about logic programs:

PROPOSITION 3. Let  $P$  be a CRWL-program and  $P_L$  its associated logic program. Then:

- (i)  $Comp(P_L) \models P_L$
- (ii) There exists a least Herbrand model  $M_{P_L}$  of  $P_L$ , which is also the least model of  $Comp(P_L)$ .
- (iii) If  $e \rightarrow t$  is ground, then  $P_L \models e \rightarrow t \Leftrightarrow M_{P_L} \models e \rightarrow t$

There is a close relation between a CWRL-program  $P$  and its associated  $P_L$ , given by the following result:

PROPOSITION 4. Let  $P$  be a CRWL-program and  $P_L$  its corresponding logic program. Then, for any expression  $e$  and term  $t$ ,

- (i)  $P_L \models e \rightarrow t \Leftrightarrow P \vdash_{CRWL} e \rightarrow t$ .
- (ii)  $Comp(P_L) \models e \rightarrow t \Rightarrow P \not\vdash_{CRWL} e \rightarrow t$  (where  $e \not\rightarrow t$  stands for  $\neg(e \rightarrow t)$ ).

PROOF. (i) We know that  $P_L \models e \rightarrow t \Leftrightarrow e \rightarrow t \in M_{P_L}$ . To prove that  $e \rightarrow t \in M_{P_L} \Leftrightarrow P \vdash_{CRWL} e \rightarrow t$ , we reason by induction on the derivation structure of  $e \rightarrow t$  in CRWL. We distinguish cases for the expression  $e$ :

- (a) When  $e = \perp$  then the only applicable rule is **BT**. Therefore  $P \vdash_{CRWL} \perp \rightarrow t \Leftrightarrow t = \perp$ , so  $P \vdash_{CRWL} \perp \rightarrow \perp \Leftrightarrow \perp \rightarrow \perp \in M_{P_L}$
- (b) Let  $e = c(e_1, \dots, e_n)$  for some  $c \in CS$ .  $P \vdash_{CRWL} c(e_1, \dots, e_n) \rightarrow t \Leftrightarrow t = \perp$  or  $t = c(t_1, \dots, t_n)$  where  $P \vdash_{CRWL} e_i \rightarrow t_i$  for every  $i$ . For the first case  $P \vdash_{CRWL} c(e_1, \dots, e_n) \rightarrow \perp \Leftrightarrow c(e_1, \dots, e_n) \rightarrow \perp \in M_{P_L}$ . For the second one by induction hypothesis  $e_i \rightarrow t_i \in M_{P_L}$  for every  $i$  and applying the third clause of the logic program  $P \vdash_{CRWL} c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n) \Leftrightarrow c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n) \in M_{P_L}$
- (c) Let  $e = f(e_1, \dots, e_n)$  for some  $f \in FS$ .  $P \vdash_{CRWL} f(e_1, \dots, e_n) \rightarrow t \Leftrightarrow t = \perp$  or there exists an instance of a program rule  $f(t_1, \dots, t_n) = r$  such that  $e_i \rightarrow t_i$  for every  $i$  and  $r \rightarrow t$ . When  $t = \perp$  the rule applied is **BT** and as in the other cases we obtain the result. If  $t$  comes from an application of a program rule then the derivation rule applied is **FR**. By induction hypothesis  $e_i \rightarrow t_i \in M_{P_L}$  for every  $i$  and  $r \rightarrow t \in M_{P_L}$ . Therefore there exists a rule  $\mathcal{R}_j$  and a substitution  $\theta$  such that  $t_i^j \theta = t_i$  for every  $i$  and  $r^j \theta = r$ . For every  $i$ ,  $t_i$  is a ground instance of  $t_i^j$ , then using the fifth clause of the logic program we obtain  $P \vdash_{CRWL} f(e_1, \dots, e_n) \rightarrow t \Leftrightarrow f(e_1, \dots, e_n) \rightarrow t \in M_{P_L}$

(ii) We reason by contradiction. Assume  $P \vdash_{CRWL} e \rightarrow t$ . By part (i),  $P_L \models e \rightarrow t$ , and therefore  $Comp(P_L) \models e \rightarrow t$ . Since  $Comp(P_L)$  is consistent ( $M_{P_L}$  is a model), we conclude  $Comp(P_L) \not\models e \not\rightarrow t$ .  $\square$

We are interested in properties which are expressible as FOL formulas  $\varphi$  over the relation  $\rightarrow$ . In this sense, we consider the following FOL theories:

$$\begin{aligned} T_{P_L} &= \{\varphi \mid P_L \models \varphi\} \\ T_{Comp(P_L)} &= \{\varphi \mid Comp(P_L) \models \varphi\} \\ T_{M_P} &= \{\varphi \mid M_P \models \varphi\} \end{aligned}$$

We are mainly interested in the properties valid in  $M_{P_L}$ , that is, in  $T_{M_P}$ . But since  $M_{P_L}$  is a model of  $P_L$  and  $Comp(P_L)$ , we have  $T_{P_L} \subseteq T_{Comp(P_L)} \subseteq T_{M_P}$ , which means that in practice we can use  $P_L$  or  $Comp(P_L)$  to obtain by FOL deduction properties of  $M_{P_L}$ .

Of course,  $T_{P_L}$  is a rather poor approximation to  $T_{M_P}$ . We find in  $T_{Comp(P_L)}$  more interesting properties, in particular related to impossible reductions from a given expression. For instance, in the *Coin* example we have  $Comp(Coin_L) \models double(coin) \not\rightarrow s(0)$ , where  $e \not\rightarrow t$  stands for  $\neg(e \rightarrow t)$ .

There are nevertheless many interesting properties of  $M_{P_L}$  which are not deducible from  $Comp(P_L)$ , in particular many inductive properties. In order to cope with (some of) these properties within the framework of FOL deduction, we consider the *inductive extension* of the completion.

*Definition 1.* Let  $P$  be a CRWL program and consider its completion  $Comp(P_L)$ . The inductive extension of the

completion,  $CompInd(P_L)$ , results of adding to  $Comp(P_L)$  the following axioms for the structural induction scheme:

For every formula  $\varphi$  with one free variable:

$$\begin{aligned} & \dots \wedge \varphi(a) \wedge \dots \wedge \varphi(g) \wedge \dots \wedge \\ & \dots \wedge \forall x_1, \dots, x_n. (\varphi(x_1) \wedge \dots \wedge \varphi(x_n) \Rightarrow \varphi(c(\bar{x}))) \wedge \dots \wedge \\ & \dots \wedge \forall x_1, \dots, x_n. (\varphi(x_1) \wedge \dots \wedge \varphi(x_n) \Rightarrow \varphi(f(\bar{x}))) \wedge \dots \\ & \Rightarrow \forall x. \varphi(x) \end{aligned}$$

where  $a, g$  range over  $CS^0$  and  $FS^0$ , and  $c, f$  over  $CS^n$  and  $FS^n$  ( $n > 0$ ).

All these FOL axioms for induction are valid in  $M_{P_L}$ , and then  $M_{P_L} \models CompInd(P_L)$ .  $CompInd(P_L)$  is powerful enough for proving many interesting properties of  $M_{P_L}$ . One example of formula valid in  $M_{P_L}$  that can be proved from  $CompInd(P_L)$  but not from  $Comp(P_L)$  is the above mentioned formula  $\forall E, T. (E \rightarrow T \Rightarrow E + 0 \rightarrow T)$ .

Let us discuss now how good is  $CompInd(P_L)$  as axiomatization of  $M_{P_L}$ . If we call  $T_{CompInd(P_L)} = \{\varphi \mid CompInd(P_L) \models \varphi\}$ , we have the following chain of FOL theories:

$$T_{P_L} \subseteq T_{Comp(P_L)} \subseteq T_{CompInd(P_L)} \subseteq T_{M_P}$$

where we know that the first two inclusions are strict. It is easy to give examples showing that also  $T_{CompInd(P_L)} \subseteq T_{M_P}$  is a strict inclusion (we start Sect. 5 with some of such examples). But note that this is an old known limitation of formalizations which comes back to Gödel incompleteness results. Since  $P_L$ ,  $Comp(P_L)$  and  $CompInd(P_L)$  are recursive,  $T_{P_L}$ ,  $T_{Comp(P_L)}$  and  $T_{CompInd(P_L)}$  are all recursively enumerable, while  $T_{M_P}$  is not, except for some very simple  $P$ .

## 4. TRANSLATION INTO SOME EXISTING FRAMEWORKS

In this section we put in practice the ideas introduced in the last section: to prove properties of the initial model of a  $CRWL$ -program, use the inductive extension of the completion of its associated logic program, and perform FOL deduction.

To this purpose, we have translated into several existing interactive proof assistants the inductive extension of the completion of some  $CRWL$ -programs. Actually, since all the used systems include induction as a built-in reasoning mechanism, it suffices to translate the completion.

To guide the discussion in this section, we use in all cases the program *Coin* in Fig. 1. and consider for it the following very simple properties:

- (P<sub>1</sub>)  $double(coin) \rightarrow 0$ : This formula is in fact a consequence of  $Coin_L$ .
- (P<sub>2</sub>)  $double(coin) \leftrightarrow s(0)$ : This formula is in fact a consequence of  $Comp(P_L)$ .
- (P<sub>3</sub>)  $\forall X, Y, T. (term(X) \wedge term(Y) \wedge X + Y \rightarrow T \Rightarrow Y + X \rightarrow T)$ : This is an inductive property deducible from  $CompInd(P_L)$ , but not from  $Comp(P_L)$ . We make use of the auxiliary predicate *term* – defined in the natural way – to recognize if an expression is indeed a constructor term.

We have used ITP [10], LPTP [30] and Isabelle [22] as proof assistants. Different reasons are behind the choice of each one of these systems: our interest in ITP is explained by the relative proximity (see [25]) of  $CRWL$  and rewriting

logic [21], the underlying logic of ITP; we expect LPTP to be useful for our purposes, because we translate  $CRWL$  into logic programming and LPTP is a specific tool for proving properties of logic programs; finally, Isabelle is a general purpose and widely used powerful proof assistant.

**The ITP prover [10]:** The ITP tool is designed to prove properties of the initial model of an equational specification written in Maude [9]. As has been explained from the very beginning in this work, it would be unsound to introduce in ITP a  $CRWL$  program as an equational specification, because of the semantics of  $CRWL$ . Instead, we must specify the reduction relation  $\rightarrow$  by means of equations giving the value *true* or *false*. In figure 2 part of this specification is shown. As can be seen, the possible reductions are split by the rules that can be applied at this moment. The condition in the rules giving the value *false* is, in consequence, the negation of the disjunction of the conditions of the rules giving *true*. To specify universal quantification we need to use new constants, which are denoted as  $C*$ .

Using this specification we obtain a perfect control on the nondeterministic reduction possibilities and therefore on the call-time choice semantics, but there is also a loss of automation when using the theorem prover tool.

We have tested this system with the three simple properties already mentioned. The property **P<sub>1</sub>** is easily proved using this tool, but not automatically, as one would desire. This is because we need to make explicit which of the possible reductions of *coin* is adequate to instantiate the existential variable **T1** which appears in the rule for *double*. In ITP, in general, rules having new variables on their right hand side cannot be applied automatically, and the user must apply the rule manually by making explicit the rule instance which is interesting to apply. When dealing with negative properties like **P<sub>2</sub>**, it is needed an application of a rule for false reductions. Such rules cannot be applied neither automatically nor manually because of the introduction of the variables  $C*$ . Therefore, we need to prove lemmas specifying the condition with universally quantified variables. Many of this lemmas introduce numerous impossible cases increasing the length of the proof. Non-determinism of the reductions of expressions bring supplementary complexity because all possible ways to obtain the result are explored. Large proofs like that of **P<sub>3</sub>** evolve into a chain of implications. This chain of implications is not directly treated as the tool does not have methods for reasoning on logical formulas. For example, to prove  $e \rightarrow t = true \Rightarrow e' \rightarrow t' = true$  we do not simplify  $e \rightarrow t$  to  $e' \rightarrow t'$  because this cannot be done by any rewriting rule. Therefore we split the proof into two different modules, one using  $e' \rightarrow t'$  and another using  $e' \leftrightarrow t'$ . The first one is the original one adding the implication step as assumption and therefore simulating the next step of the chain of implications. For the second one we have to prove, using a new lemma, the impossibility of such an assumption. When reasoning on the chain of implications we also introduce many negative proofs increasing the complexity. The following steps of the proof are not automatic because they use internal assumptions of the module.

**The LPTP prover [30]:** LPTP is a theorem prover for success, failure and termination properties of Prolog programs. To use this tool we only have to translate a logic program expressing  $CRWL$  properties into a Prolog program.

```

op _->_ : Expression Expression -> Bool .
op _+_ : Expression Expression -> Expression [ctor] .
op double : Expression -> Expression [ctor] .
op coin : -> Expression [ctor] .
...
ceq (X + Y) -> T = true if eq(T, bottom) [label sumbot] .
ceq (X + Y) -> T = true if ((X -> 0) and (Y -> T)) .
ceq (X + Y) -> T = true if ((X -> s(T1)) and (s(T1 + Y) -> T)) [label sumI] .
ceq (X + Y) -> T = false if ((not eq(T, bottom)) and (not ((X -> 0) and (Y -> T)))
and (not ((X -> s(Z*)) and ((s(Z* + Y) -> T)))))) [label redmas] .

ceq double(X) -> T = true if eq(T, bottom) [label doublebot] .
ceq double(X) -> T = true if ((X -> T1) and ((T1 + T1) -> T)) [label pdob] .
ceq double(X) -> T = false if ((not eq(T, bottom)) and (not ((X -> Y*)
and ((Y* + Y*) -> T)))) [label nredd] .

ceq coin -> T = true if eq(T, bottom) .
ceq coin -> T = true if (0 -> T) .
ceq coin -> T = true if (s(0) -> T) .
ceq coin -> T = false if ((not eq(T, bottom)) and (not(0 -> T)) and (not (s(0) -> T))) .
...

```

Figure 2: Part of Maude specification for *Coin*

LPTP automatically generates the inductive completion of the program. One of the advantages of using this tool is that, being LPTP a prover for Prolog properties, the introduction of non-determinism does not cause as many problems as in ITP. Therefore, proving  $\mathbf{P}_1$  is simpler with LPTP.

Testing the second and third properties LPTP has as many problems as ITP. First, there are too many possibilities in the reduction relation for negative or universally quantified properties. Second, the proof simplifies the goal adding the corresponding assumptions to the theory. This causes a growth on the number of variables. For properties as simple as those introduced here the system generates a complex proof of more than a thousand lines.

**Isabelle [22]:** Isabelle/HOL is a theorem prover where specifications and validations are considered on Higher-Order logic. In this case we specify the system as an inductive set for the least model of the logic program. In such a least model we can prove positive and negative facts about the reduction relation and also inductive properties of it. In figure 3 appears part of the theory on which the results are proved.

Isabelle provides methods to reason on logic formulas, relations and sets. Using these methods the property  $\mathbf{P}_1$  was proved automatically. Negative properties like  $\mathbf{P}_2$  require reasoning on the completion. This can be done using axioms for inductive sets. Similarly as in the other systems, the different ways to derive the same term in *CRWL* introduce many repeated facts to be proved. On the other hand it is not difficult to prove known facts of this calculus such as transitivity of the reduction relation. Inductive properties like  $\mathbf{P}_3$  can be expressed by a first order logic formula, then applying the rules for such formulas it is not difficult to prove the property. This translation does not introduce limitations on the formulas that can be specified nor on the induction mechanisms.

## 4.1 Improving determinism of *CRWL*

A common problem arising in the three approaches is the repetition of essentially the same proofs. The problem comes from the source logic *CRWL*. For a constructor term  $t$ , *CRWL* provides many different approximations  $t \rightarrow t'$ , for all  $t' \sqsubseteq t$ , that is, for all different  $t'$  obtained by replacing some subterms of  $t$  by  $\perp$ . This kind of non-determinism of  $\rightarrow$  can be avoided, since for constructor terms  $t$ , only the maximal approximation  $t \rightarrow t$  is really necessary. In this section we present a simplified *CRWL* calculus eliminating all those superfluous reductions associated to terms.

*Definition 2.* The proof calculus *CRWL'* results of replacing the rule (*BT*) in *CRWL* (Fig. 1) by the new rule (*BT'*)

$$\frac{}{e \rightarrow \perp} \quad \text{if } e = f(e_1, \dots, e_n) \text{ or } e = \perp$$

The next result relates the provable statements of *CRWL* and *CRWL'*.

**PROPOSITION 5.** *Let  $P$  be a *CRWL*-program. For any expression  $e$  and term  $t$ :*

- (i)  $P \vdash_{CRWL} e \rightarrow t \Rightarrow P \vdash_{CRWL'} e \rightarrow t'$  for some  $t' \sqsupseteq t$ .
- (ii)  $P \vdash_{CRWL'} e \rightarrow t \Rightarrow P \vdash_{CRWL} e \rightarrow t$

*As a consequence, if  $t$  is a total term:  $P \vdash_{CRWL} e \rightarrow t \Leftrightarrow P \vdash_{CRWL'} e \rightarrow t$*

**PROOF.** For the sake of readability, we will simply write  $e \rightarrow t$  instead of  $P \vdash_{CRWL} e \rightarrow t$  and  $e \rightarrow' t$  instead of  $P \vdash_{CRWL'} e \rightarrow t$ .

- (i) We reason on the size of the proof of  $e \rightarrow t$ :
  - (a) *size* = 1: We can apply the *CRWL* rule **BT**, obtaining  $e \rightarrow \perp$ . For every expression  $e$ , it is not difficult to see by induction that there is some  $t'$

```

theory Arrows = Main:

datatype exp = bottom | zero | s exp | coin | sum exp exp | double exp

consts arrow :: "(exp * exp) set"
inductive arrow
intros
bt [intro]: "(x, bottom) : arrow"
dczero [intro]: "(zero, zero) : arrow"
dcs [intro]: "(x, t):arrow ==> (s x, s t):arrow"
fcoin1 [intro]: "(zero, t):arrow ==> (coin, t):arrow"
fcoin2 [intro]: "(s(zero), t):arrow ==> (coin, t):arrow"
sum1 [intro]: "[|(x, zero):arrow ; (y, t):arrow|] ==> (sum x y,t):arrow"
sum2 [intro]: "[|(x, s(t1)):arrow ; (y,t2):arrow ; (s(sum t1 t2),t):arrow|]
==> (sum x y , t):arrow"
double [intro]: "[|(x, t1):arrow ; (sum t1 t1,t):arrow|] ==> (double(x), t):arrow"
...

```

Figure 3: Part of Isabelle specification for *Coin*

such that  $e \rightarrow' t'$ . By definition of the relation  $\sqsubseteq$ ,  $t' \sqsupseteq \perp$ . When we apply the *CRWL* rule **DC** for  $c \in CS^0$ , we obtain  $c \rightarrow c$  and we have  $c \rightarrow' c$ .

- (b)  $size > 1$ : The rules we could have applied are **DC** or **FR**. In case we have applied the rule **DC**, then  $e = c(e_1, \dots, e_n)$  and  $t = c(t_1, \dots, t_n)$ . By definition of rule **DC** there are proofs in *CRWL* for  $e_i \rightarrow t_i$  for every  $i$  and, therefore, by induction hypothesis  $e_i \rightarrow t'_i$  for some  $t'_i$  such that  $t'_i \sqsupseteq t_i$  for every  $i$ . Thus, applying the second *CRWL*' rule  $c(e_1, \dots, e_n) \rightarrow c(t'_1, \dots, t'_n)$  with  $c(t'_1, \dots, t'_n) \sqsupseteq c(t_1, \dots, t_n)$  is obtained.

Finally we can apply **FR**. In this case we obtain  $e = f(e_1, \dots, e_n)$  with proofs for  $e_i \rightarrow t_i$  for every  $i$  and  $r \rightarrow t$ , where  $f(t_1, \dots, t_n) \rightarrow r$  is an instance of a program rule  $f(t'_1, \dots, t'_n) \rightarrow r''$ . By induction hypothesis, there are  $t'_i \sqsupseteq t_i$  for every  $i$  and  $t' \sqsupseteq t$  such that  $e_i \rightarrow' t'_i$  for every  $i$ . Since  $f(t_1, \dots, t_n) \rightarrow r$  is an instance of  $f(t''_1, \dots, t''_n) \rightarrow r''$ , there must exist a substitution  $\theta$  such that  $t_i = t''_i \theta$  for every  $i$  and  $r = r'' \theta$ . The substitution  $\theta$  associates a subterm  $st_i$  of  $t_i$  to every variable of  $t''_i$ . Considering the subterm  $st'_i$  of  $t'_i$  located at the same nested level than  $st_i$  and by the linearity of the rules of a *CRWL* program, we obtain a substitution  $\theta'$  associating to each variable of  $t''_i$  the subterm  $st'_i$ . Then, as  $t'_i \sqsupseteq t_i$ , by the definition of  $\sqsubseteq$ ,  $st'_i \sqsupseteq st_i$  for every  $i$  and therefore  $\theta' \sqsupseteq \theta$ . Applying the substitution  $\theta'$ ,  $t'_i = t''_i \theta'$  for every  $i$  and  $r' = r'' \theta'$ . Therefore  $f(t'_1, \dots, t'_n) \rightarrow r'$  is an instance of the rule  $f(t''_1, \dots, t''_n) \rightarrow r''$ . By a monotonicity result of *CRWL*[13],  $r \rightarrow t \Rightarrow r' \rightarrow t$ . Finally, by the induction hypothesis, there is some  $t' \sqsupseteq t$  such that  $r' \rightarrow' t'$ . Applying the *CRWL*' third rule with  $t', t'_1, \dots, t'_n$  we obtain the result.

(ii) We reason by induction on the size of the proof  $e \rightarrow' t$ :

- (a)  $size = 1$ : Suppose we have applied the first *CRWL*' rule. In this case  $t = \perp$  and applying the *CRWL* rule **BT** we reach the result. The other possibil-

ity is the application of the second rule for some  $c \in CS^0$  and then  $c \rightarrow' c$ . In this case we also have  $c \rightarrow c$  by applying the *CRWL* rule **DC**.

- (b)  $size > 1$ : There are two possible rules that can be applied. When the second rule is applied,  $e = c(e_1, \dots, e_n)$  and  $t = c(t_1, \dots, t_n)$  and there are proofs for  $e_i \rightarrow' t_i$  for every  $i$ . By induction hypothesis  $e_i \rightarrow t_i$  for every  $i$  and now applying the *CRWL* rule **DC**,  $e \rightarrow t$  is obtained.

If the third rule is applied then  $e = f(e_1, \dots, e_n)$  for some  $f \in FS$  and there are proofs for  $e_i \rightarrow' t_i$  for every  $i$  and  $r \rightarrow' t$  if  $f(t_1, \dots, t_n) \rightarrow r$  is a partial-instance of a program rule. Again by induction hypothesis  $e_i \rightarrow t_i$  for every  $i$  and  $r \rightarrow t$  and applying the *CRWL* rule **FR**,  $e \rightarrow t$  is obtained.  $\square$

We have tested our sample properties with the refined calculus *CRWL*', conveniently translated to the different systems, obtaining significant shortenings in the proofs. Furthermore, since reduction between c-terms is now deterministic, it is possible to use equational reasoning in those parts of the proofs involving this kind of reductions. This has been a further source of simplification of the proofs while using ITP, because it gives more chances to automation.

## 5. BEYOND THE COMPLETION: AXIOMATIZING DERIVABILITY

As we discussed at the end of Sect. 3, no FOL axiomatization can be complete for the least model of a program. In the case of  $CompInd(P_L)$ , although it covers many interesting properties, it is nevertheless quite easy to find examples revealing its limitations. Consider for example the following simple program *Loop*:

`loop  $\rightarrow$  loop`

It is not difficult to see that  $loop \rightarrow 0$  is valid in  $M_{Loop_L}$ , but  $CompInd(Loop_L) \not\models loop \rightarrow 0$ . A less trivial example is given by the following program *Even*:

$\text{even}(0) \rightarrow \text{true}$              $\text{an\_even} \rightarrow 0$   
 $\text{even}(s(0)) \rightarrow \text{false}$        $\text{an\_even} \rightarrow s(s(\text{an\_even}))$   
 $\text{even}(s(s(X))) \rightarrow \text{even}(X)$

Notice that  $\text{an\_even}$  admits an infinite number of reductions giving all the even natural numbers. The property  $\text{even}(\text{an\_even}) \rightarrow \text{false}$  is valid in  $M_{\text{Even}_L}$  but, again, is not deducible from  $\text{CompInd}(\text{Even}_L)$ .

We remark that the two given examples express negative properties involving nontermination. It is not so strange that completion is not able to prove them, since it is known that completion is related to finite failure. But nontermination analysis by itself does not suffice to prove the properties. Notice also that, in both cases, the properties can be proved by inductive reasoning over the universe of  $\text{CRWL}$ -derivations. This suggests some meta-theory at the object level, by considering a variant of  $\text{CRWL}$  (to be precise, of the logic program mirroring  $\text{CRWL}$ ) where the  $\text{CRWL}$ -derivation trees for statements  $e \rightarrow t$  are made explicit.

We first introduce some constructor terms representing  $\text{CRWL}$ -derivations.

*Definition 3.* The set of *derivation constructors symbols*  $\text{CSDer}$  consists of the following symbols:

$bt \in \text{CSDer}^0$   
 $dc_c \in \text{CSDer}^k$  for every  $c \in \text{CS}^k$   
 $fa_{f,R} \in \text{CSDer}^{k+1}$  for every  $R$  rule for  $f$  and  $f \in \text{FS}^k$ .

Constructor terms built up with derivation constructors are called *derivation terms*.

We will use  $d, d', \dots$  to denote derivation terms.

Now, given a  $\text{CRWL}$ -program  $P$ , we associate to it a logic program defining a ternary relation  $d \vdash e \rightarrow t$  whose intended meaning is ‘ $d$  represents a  $\text{CRWL}$ -derivation of  $e \rightarrow t$ ’.

*Definition 4.* Given a  $\text{CRWL}$  program  $P$  the associated logic program making explicit the proofs,  $\text{Der}(P)$ , consists of the following clauses defining  $\_ \vdash \_ \rightarrow \_$  as a ternary relation:

$bt \vdash \perp \rightarrow \perp$   
 For every  $c \in \text{CS}$ :  
 $bt \vdash c(E_1, \dots, E_n) \rightarrow \perp$   
 $dc_c(D_1, \dots, D_n) \vdash c(E_1, \dots, E_n) \rightarrow c(T_1, \dots, T_n) \Leftarrow$   
 $D_1 \vdash E_1 \rightarrow T_1 \wedge \dots \wedge D_n \vdash E_n \rightarrow T_n$   
 For every  $f \in \text{FS}$ :  
 $bt \vdash f(E_1, \dots, E_n) \rightarrow \perp$   
 For every rule  $\mathcal{R}$ ,  $f(t_1, \dots, t_n) = e$ :  
 $fa_{f,\mathcal{R}}(D_1, \dots, D_n, D) \vdash f(E_1, \dots, E_n) \rightarrow T \Leftarrow$   
 $D_1 \vdash E_1 \rightarrow t_1 \wedge \dots \wedge D_n \vdash E_n \rightarrow t_n \wedge D \vdash e \rightarrow T$

As we did with  $P_L$  in Sect. 3, we can think on the least model  $M_{\text{Der}(P)}$  of  $\text{Der}(P)$ , the completion  $\text{Comp}(\text{Der}(P))$  and its inductive extension  $\text{CompInd}(\text{Der}(P))$ .

Considering the *Loop* and *Even* examples and this new approach, we have  $\text{CompInd}(\text{Der}(\text{Loop})) \models \text{loop} \rightarrow 0$  and  $\text{CompInd}(\text{Der}(\text{Even})) \models \text{even}(\text{an\_even}) \rightarrow \text{false}$ .

We explore now some logical relations between  $\text{Der}(P)$  and the original program. Our first result relates the reduction statements derived using this approach and those of the original calculus.

**PROPOSITION 6.** *For every  $P$   $\text{CRWL}$  program, and for every  $e$  expression and  $t$  term:*

- (i)  $\text{Der}(P) \models \exists D.D \vdash e \rightarrow t \Leftrightarrow P_L \models e \rightarrow t \Leftrightarrow P \vdash_{\text{CRWL}} e \rightarrow t$
- (ii)  $\text{Comp}(\text{Der}(P)) \models \nexists D.D \vdash e \rightarrow t \Rightarrow P \not\vdash_{\text{CRWL}} e \rightarrow t$

**PROOF.** (i) If we prove  $\text{Der}(P) \models \exists D.D \vdash e \rightarrow t \Leftrightarrow P \vdash_{\text{CRWL}} e \rightarrow t$  then we obtain  $\text{Der}(P) \models \exists D.D \vdash e \rightarrow t \Leftrightarrow P_L \models e \rightarrow t$  by proposition 4.

( $\Rightarrow$ ) It is easy to see that  $\text{Der}(P) \models \exists D.D \vdash e \rightarrow t \Leftrightarrow M_{\text{Der}(P)} \models d \vdash e \rightarrow t$  for some derivation term  $d$ . We now reason by induction on the structure of the expression  $e$ :

$e = \perp$ : If  $M_{\text{Der}(P)} \models d \vdash \perp \rightarrow t$  then  $d = bt$  and  $t = \perp$  and  $P \vdash_{\text{CRWL}} \perp \rightarrow \perp$  by rule **BT**.

$e = c(e_1, \dots, e_n)$ : If  $M_{\text{Der}(P)} \models d \vdash c(e_1, \dots, e_n) \rightarrow t$  then we will have two different possibilities. When  $t = \perp$  we apply the rule **BT** and then  $P \vdash_{\text{CRWL}} c(e_1, \dots, e_n) \rightarrow \perp$ . The other case is  $t = c(t_1, \dots, t_n)$  and  $d = dc_c(d_1, \dots, d_n)$  such that  $M_{\text{Der}(P)} \models d_i \vdash e_i \rightarrow t_i$  for every  $i$ . We have  $P \vdash_{\text{CRWL}} e_i \rightarrow t_i$  by I.H. and considering the rule **DC**,  $P \vdash_{\text{CRWL}} c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n)$  is obtained.

$e = f(e_1, \dots, e_n)$ : As in the other case we have two possibilities when  $M_{\text{Der}(P)} \models d \vdash f(e_1, \dots, e_n) \rightarrow t$ . For  $t = \perp$  we have the result applying the same argument as before. The other possibility is  $d = fa_{f,R_j}(d_1, \dots, d_n, d')$ , for some program rule  $R_j$ ,  $f(t_1, \dots, t_n) = r_j$ , and  $M_{\text{Der}(P)} \models d_i \vdash e_i \rightarrow t_i$  for every  $i$  and  $M_{\text{Der}(P)} \models d' \vdash r_j \rightarrow t$ . Then we have  $P \vdash_{\text{CRWL}} e_i \rightarrow t_i$  and  $P \vdash_{\text{CRWL}} r_j \rightarrow t$  by I.H. Now, applying the **FR** rule, we reach  $P \vdash_{\text{CRWL}} f(e_1, \dots, e_n) \rightarrow t$

( $\Leftarrow$ ) We reason by induction on the structure of the  $\text{CRWL}$  proof for  $e \rightarrow t$ .

**Rule BT:** Then  $t = \perp$ . For every  $e$ ,  $\text{Der}(P) \models bt \vdash e \rightarrow \perp$ . Therefore,  $\text{Der}(P) \vdash \exists D.D \vdash e \rightarrow t$ .

**Rule DC:** For this case  $P \vdash_{\text{CRWL}} c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n)$  with  $P \vdash_{\text{CRWL}} e_i \rightarrow t_i$  for every  $i$ . By I.H. we obtain  $\text{Der}(P) \models \exists D.D \vdash e_i \rightarrow t_i$  for every  $i$  and then  $M_{\text{Der}(P)} \models d_i \vdash e_i \rightarrow t_i$  for some derivation term  $d_i$ . Since  $M_{\text{Der}(P)} \models dc_c(d_1, \dots, d_n) \vdash c(e_1, \dots, e_n)$  we have  $\text{Der}(P) \models \exists D.D \vdash c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n)$ .

**Rule FR:** For some program rule  $R_j : f(t_1, \dots, t_n) = r_j$ , we have  $P \vdash_{\text{CRWL}} f(e_1, \dots, e_n) \rightarrow t$  with  $P \vdash_{\text{CRWL}} e_i \rightarrow t_i$  and  $P \vdash_{\text{CRWL}} r_j \rightarrow t$ . Then  $\text{Der}(P) \models \exists D.D \vdash e_i \rightarrow t_i$  and  $\text{Der}(P) \models \exists D.D \vdash r_j \rightarrow t$  are obtained by I.H.. Now, considering the minimal model, we have  $M_{\text{Der}(P)} \models d_i \vdash e_i \rightarrow t_i$  and  $M_{\text{Der}(P)} \models d' \vdash r_j \rightarrow t$  for some derivation terms  $d_1, \dots, d_n, d'$ . Then applying the rules of the logic program for  $\text{Der}(P)$ ,  $M_{\text{Der}(P)} \models fa_{f,R_j}(d_1, \dots, d_n, d') \vdash f(e_1, \dots, e_n) \rightarrow t$  and therefore  $\text{Der}(P) \models \exists D.D \vdash c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n)$ .

- (ii) We reason by contradiction. Assume  $P \vdash_{\text{CRWL}} e \rightarrow t$ . Then  $\text{Der}(P) \models \exists D.D \vdash e \rightarrow t$  by part (i). As  $\text{Comp}(\text{Der}(P)) \models \text{Der}(P)$  then  $\text{Comp}(\text{Der}(P)) \models \exists D.D \vdash e \rightarrow t$  holds. Since  $\text{Comp}(\text{Der}(P))$  is consistent, we conclude  $\text{Comp}(\text{Der}(P)) \not\models \exists D.D \vdash e \rightarrow t$   $\square$

In order to compare the behavior of  $\text{Der}(P)$  with respect to more general properties  $\varphi$ , we define a natural conversion

of FOL formulas using the relation  $\_ \rightarrow \_$  into formulas using  $\_ \vdash \_ \rightarrow \_$ , as well as a natural relation between models of  $P_L$  and of  $Der(P)$ .

*Definition 5.* (i) If  $\varphi$  is a FOL formula using the relation  $\_ \rightarrow \_$ , we write  $\widehat{\varphi}$  for the result of replacing in  $\varphi$  each subformula  $e \rightarrow t$  by  $\exists D.D \vdash e \rightarrow t$ .

(ii) Let  $M$  be a model for  $P_L$ , we define the following set  $S_M$  of models of  $Der(P)$ :

$S_M = \{M' \models Der(P) \mid \forall e, t (M \models e \rightarrow t \Leftrightarrow \text{exists } d \text{ such that } M' \models d \vdash e \rightarrow t)\}$

PROPOSITION 7. (i)  $M' \models Der(P)$  iff there exists  $M \models P_L$  such that  $M' \in S_M$

(ii)  $M_{Der(P)} \in S_{M_{P_L}}$

PROOF. (i)  $(\Rightarrow)$ : Given a model  $M'$  for  $Der(P)$  we can construct  $M$  considering only those  $e \rightarrow t$  that correspond to some  $d \vdash e \rightarrow t$  valid in  $M'$ . By construction the only requirement that needs to be checked is that such  $M$  is a model of  $P_L$ . Therefore we have to prove  $M \models \varphi$  for every  $\varphi \in P_L$ . We examine each clause in the definition of  $P_L$  given in section 3.

(a) As  $M'$  is model for  $Der(P)$  then  $bt \vdash \perp \rightarrow \perp \in M'$ . Therefore, by construction,  $\perp \rightarrow \perp \in M$

(b)  $M'$  model of  $Der(P) \Rightarrow bt \vdash c(e_1, \dots, e_n) \rightarrow \perp \in M'$ . Then by construction of  $M$ ,  $c(e_1, \dots, e_n) \rightarrow \perp \in M$

(c) Suppose  $e_i \rightarrow t_i \in M$ , we have to prove  $c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n) \in M$ . As  $e_i \rightarrow t_i \in M$ , then there exists  $d_i$  such that  $d_i \vdash e_i \rightarrow t_i$ .  $M'$  is a model for  $Der(P)$ , then  $dc_c(d_1, \dots, d_n) \vdash c(e_1 \rightarrow t_1) \rightarrow c(t_1, \dots, t_n) \in M'$ . By construction of  $M$ ,  $c(e_1, \dots, e_n) \rightarrow c(t_1, \dots, t_n) \in M$  is obtained.

(d) Similar to (b)

(e) Similar to (c)

$(\Leftarrow)$ : This is trivial by the definition of  $S_M$

(ii) By definition of  $S_M$ , we have  $M_{Der(P)} \in S_{M_{P_L}}$  iff  $d \vdash e \rightarrow t \in M_{Der(P)} \Leftrightarrow e \rightarrow t \in M_{P_L}$ . Assume  $d \vdash e \rightarrow t \in M_{Der(P)}$ , this is equivalent to  $Der(P) \models \exists D.D \vdash e \rightarrow t$ . Then,  $Der(P) \models \exists D.D \vdash e \rightarrow t \Leftrightarrow P_L \models e \rightarrow t$  follows by proposition 6. Therefore, as  $e \rightarrow t$  is an atomic formula,  $P_L \models e \rightarrow t \Leftrightarrow e \rightarrow t \in M_{P_L}$   $\square$

The following result relates validity in a model of  $P_L$  with validity in the corresponding models of  $Der(P)$

PROPOSITION 8. Let  $\varphi$  be a formula and  $M$  model of  $P_L$  and  $M' \in S_M$  then:

$$M \models \varphi \Leftrightarrow M' \models \widehat{\varphi}$$

PROOF. We reason by structural induction on the FOL formula  $\varphi$

Case  $\varphi$  atomic: The atomic formulas are of the form  $e \rightarrow t$ . Considering  $M \models e \rightarrow t$ , then by definition of  $S_M$ , there exists a derivation term  $d$  such that  $d \vdash e \rightarrow t \in M'$ . Therefore  $M' \models \exists D.D \vdash e \rightarrow t$ . The other direction of the implication is shown in a similar way. If  $M' \models \exists D.D \vdash e \rightarrow t$  then there exists  $d$  such that  $d \vdash e \rightarrow t \in M'$ . Therefore by definition of  $S_M$  we obtain  $M \models e \rightarrow t$ .

Case  $\varphi = \neg\varphi_1$ :  $M \models \neg\varphi_1 \Leftrightarrow M \not\models \varphi_1 \Leftrightarrow$  (by i.h.)  $M' \not\models \widehat{\varphi_1} \Leftrightarrow M' \models \neg\widehat{\varphi_1}$  and  $\neg\widehat{\varphi_1} = \widehat{\neg\varphi_1}$

Case  $\varphi = \varphi_1 \wedge \varphi_2$ :  $M \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M \models \varphi_1$  and  $M \models \varphi_2 \Leftrightarrow$  (by i.h.)  $M' \models \widehat{\varphi_1}$  and  $M' \models \widehat{\varphi_2} \Leftrightarrow M' \models \widehat{\varphi_1} \wedge \widehat{\varphi_2}$  and  $\widehat{\varphi_1} \wedge \widehat{\varphi_2} = \widehat{\varphi_1 \wedge \varphi_2}$ .

Case  $\varphi = \exists x.\varphi_1$ :  $M \models \exists x.\varphi_1 \Leftrightarrow$  exists  $c \in T_\Sigma$  such that  $M \models \varphi_1[c/x] \Leftrightarrow$  (by i.h.) exists  $c \in T_\Sigma$   $M' \models \widehat{\varphi_1[c/x]}$  and  $\widehat{\varphi_1[c/x]} = \widehat{\varphi_1}[c/x] \Leftrightarrow M' \models \exists x.\widehat{\varphi_1}$  and  $\exists x.\widehat{\varphi_1} = \widehat{\exists x.\varphi_1}$   $\square$

As a consequence of the previous results, we conclude also that the properties derived from  $P_L$  and from  $Der(P)$  are the same (via  $\widehat{\phantom{x}}$ ), as stated by the following proposition:

PROPOSITION 9. For any  $\varphi$ ,  $P_L \models \varphi \Leftrightarrow Der(P) \models \widehat{\varphi}$ .

PROOF. It is a consequence of propositions 7 and 8  $\square$

All these results show that nothing new can be obtained from  $Der(P)$  and  $M_{Der(P)}$  with respect to  $P_L$  and  $M_{P_L}$ . The *Loop* and *Even* examples show that the real gain comes from  $CompInd(Der(P))$  with respect to  $CompInd(P_L)$ . Therefore, those properties not involving reasoning on the structure of the CRWL-derivation will be proved using the first approach, where the proofs are simpler. Only when reasoning on the structure of the derivation is needed the second approach will be used.

We have tested this new approach with ITP and Isabelle. As was expected, with this approach we can prove properties reasoning by structural induction on the derivation terms. As an example, consider the program *Loop*. Its associated translation into Isabelle is shown in figure 4. It is not too difficult to prove  $loop \rightarrow 0$  reasoning by induction on the derivations and discarding all those incorrect derivations. The proof is slightly complicated because of the introduction of such incorrect cases, but the steps are not difficult.

As has been previously remarked, the resulting proofs with the new approach can be in general more complicated than the corresponding ones with the original approach, whenever the latter is applicable. But this is not always true. For instance, consider again the program *Coin* and the sample properties of section 4. The property  $\mathbf{P}_1$ , rephrased as  $\exists D.D \vdash coin \rightarrow 0$ , can be still proved automatically in Isabelle. The situation is different for negative properties like  $\mathbf{P}_2$ , that are expressed in the new approach as universal quantifications over derivations. Therefore, when trying to prove such negative properties we have to inspect all possible derivations. There are only a few of them possible for a given expression as can be deduced from the logic program  $Der_P$ , but all the possibilities have to be explored, hence complicating the proof.

## 6. CONCLUSIONS

We have presented some logical conceptual tools for proving properties of first order functional logic programs. Programs consist of constructor based rewrite systems possibly non-terminating and non-confluent, defining thus non-strict non-deterministic functions, with call-time choice semantics. This corresponds to the first order core of existing modern FLP systems like Curry or Toy.

Our logical starting point has been *CRWL*, a well known semantic framework for FLP. *CRWL* includes a proof calculus giving logical semantics to programs, and a model theory satisfying that every program has an initial model. The

```

theory Demos = Main:

datatype exp = bottom | zero | s exp | coin | sum exp exp | double exp | loop
datatype dem = bt | dczero | dcs dem | facoin1 dem | facoin2 dem | fasum1 dem dem
             | fasum2 dem dem dem | fadouble dem dem | faloop dem

consts demo :: "(dem * exp * exp) set"
inductive demo
intros
rbt [intro]: "(bt, x, bottom) : demo"
rdczero [intro]: "(dczero, zero, zero) : demo"
rdcs [intro]: "(d, x, t):demo ==> (dcs d, s x, s t):demo"
rfcoin1 [intro]: "(d, zero, t):demo ==> (facoin1 d, coin, t):demo"
rfcoin2 [intro]: "(d, s(zero), t):demo ==> (facoin2 d, coin, t):demo"
rsum1 [intro]: "[|(d, x, zero):demo ; (d1, y, t):demo|]
               ==> (fasum1 d d1, sum x y, t):demo"
rsum2 [intro]: "[|(d, x, s(t1)):demo ; (d1, y,t2):demo ; (d2, s(sum t1 t2), t):demo|]
               ==> (fasum2 d d1 d2, sum x y , t):demo"
rdouble [intro]: "[|(d, x, t1):demo ; (d1, sum t1 t1,t):demo|]
                 ==> (fadouble d d1, double(x), t):demo"
rloop [intro]: "(d, loop, t):demo ==> (faloop d, loop, t):demo"

```

Figure 4: Isabelle specification of the least model of  $Der(P)$

program properties of interest are those valid in that initial model, which are then typically inductive properties.

In order to prove such program properties, we have mapped *CRWL* into logic programming in the following sense: to each *CRWL*-program  $P$  we associate in a simple manner a logic program  $P_L$  such that the least model of  $P_L$  consists exactly of the reduction statements which are *CRWL*-provable from  $P$ . As a nice consequence, all the machinery (theoretical and practical) of logic programming is available to us. For instance, the completion of  $P_L$  can be used to deduce negative results, and with its inductive extension we can deduce inductive properties of the least model.

We have made experiments with this approach by encoding into several existing proof assistants the completion of simple programs (the inductive extension is implicit in all these systems). Namely, we have used: ITP [10], a tool based on rewriting logic [21] and designed for proving properties of equational specifications; LPTP [30], a tool designed specifically for logic programs; and Isabelle [22], a well known general purpose proof assistant. In all cases, to prove simple properties of *CRWL*-programs is not as easy as one would desire. We have detected two particular aspects having great impact in the simplicity of proofs. One is, of course, the concrete encoding: for instance, in the ITP case, a sorted version (distinguishing terms and expressions) was clearly better than an unsorted one. The other one is the formulation of the *CRWL* logic itself: we have proposed a refinement eliminating superfluous sources of non-determinism of the reduction relation  $\rightarrow$ , with which some proofs are remarkably simpler and shorter.

Of course, due to Gödel-like arguments, no deductive system can prove all properties of initial models. The limits of the completion+induction approach are easily reachable by considering properties which are valid due to non-termination. This is natural, since completion is closely related to finite failure.

To enlarge the class of provable properties we have then sophisticated the logic programming specification  $P_L$  of the

semantics of a *CRWL*-program  $P$ , by making explicit the *CRWL*-proof tree corresponding to *CRWL*-provable reduction statements for  $P$ . The resulting logic program  $Der(P)$  has its own completion  $Comp(Der(P))$ , inductive extension of the completion  $CompInd(Der(P))$ , and its least model  $M_{Der(P)}$ . An interesting point is that the logical consequences of  $Der(P)$  and  $Comp(Der(P))$  are essentially the same of  $P_L$  and  $Comp(P_L)$ , and the same happens with the valid properties in  $M_{P_L}$  and  $M_{Der(P)}$ . What produces new results is  $CompInd(Der(P))$  with respect to  $CompInd(P_L)$ , as we have indeed shown in our implementations.

We have in mind many things to do as future work. In the practical side it is important to test the approach with interesting non trivial case studies. In the theoretical side we plan to improve the approach by making the mapping of logics more precise, refining the target logic by considering many sorted logic programs, and refining the source logic by considering extensions of *CRWL* with other features like HO [14, 15] or failure [20].

## 7. REFERENCES

- [1] M. Alpuente, F.J. Correa, M. Falaschi. *A Debugging Scheme of Functional Logic Programs*, Proc. WFLP'01, Electronic Notes on Theoretical Computer Science, Vol 64, 2002.
- [2] M. Alpuente, D. Ballis, F.J. Correa, M. Falaschi. *Automated Correction of Functional Logic Programs*, Proc. European Symp. on Programming (ESOP'03), Springer LNCS 2618, pp. 54-68, 2003.
- [3] S. Antoy. *Optimal Non-deterministic Functional Logic Computations*, Proc. ALP/HOA 1997, Springer LNCS 1298, pp. 16-30, 1997.
- [4] S. Antoy. *Constructor-based Conditional Narrowing*. Proc. Principles and Practice of Declarative Programming (PPDP'01), 199-206, ACM Press, 2001.
- [5] K.R. Apt. *Logic Programming*. In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B, Chapter 10, Elsevier and The MIT Press, pp. 493-574, 1990.

- [6] D. Bert, R. Echahed. *Abstraction of Conditional Term Rewriting Systems*. Proc ILPS 1995 , pp. 162–176, 1995.
- [7] R. Caballero, F.J. López-Fraguas and M. Rodríguez-Artalejo. *DDT: Theoretical Foundations for the Declarative Debugging of Lazy Functional Logic Programs*. Proc. of the 5th International Symposium on Functional and Logic Programming (FLOPS'2001), Springer LNCS 2024, pp. 170–184, 2001.
- [8] R. Caballero and M. Rodríguez-Artalejo. *A Declarative Debugging System for Lazy Functional Logic Programs*. Electronic Notes in Theoretical Computer Science 64, 63 pages, 2002.
- [9] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott. *Maude 2.0 Manual*. <http://maude.cs.uiuc.edu>, 2003.
- [10] M. Clavel. *The ITP tool*. In A. Nepomuceno, J. F. Quesada, and J. Salguero, editors, Logic, Language and Information. Proc. of the 1st Workshop on Logic and Language, Kronos, 55–62, 2001. System available at <http://www.ucm.es/info/dsip/clavel/itp>
- [11] H.B. Enderton. *A Mathematical Introduction to Logic*, Academic Press, 2001.
- [12] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas and M. Rodríguez-Artalejo. *A Rewriting Logic for Declarative Programming*. Proc. European Symp. on Programming (ESOP'96), Springer LNCS 1058, pp. 156–172, 1996.
- [13] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas and M. Rodríguez-Artalejo. *An Approach to Declarative Programming Based on a Rewriting Logic*. Journal of Logic Programming 40(1), pp. 47–87, 1999.
- [14] J.C. González-Moreno, M.T. Hortalá-González and M. Rodríguez-Artalejo. *A Higher Order Rewriting Logic for Functional Logic Programming*. Proc. Int. Conf. on Logic Programming, The MIT Press, pp. 153–167, 1997.
- [15] J.C. González-Moreno, M.T. Hortalá-González and M. Rodríguez-Artalejo. *Polymorphic Types in Functional Logic Programming*. FLOPS'99 special issue of the Journal of Functional and Logic Programming, 2001. <http://danae.uni-muenster.de/lehre/kuchen/JFLP>.
- [16] M.J.C. Gordon and T.F. Melham. *Introduction to HOL*, Cambridge Univ. Press, 1993.
- [17] M. Hanus (ed.), *Curry: an Integrated Functional Logic Language*, Version 0.8, April 15, 2003. <http://www-i2.informatik.uni-kiel.de/~curry/>.
- [18] H. Hussmann. *Nondeterministic Algebraic Specifications and Nonconfluent Term Rewriting*. Journal of Logic Programming 12, pp. 237–255, 1992.
- [19] F.J. López-Fraguas, J. Sánchez Hernández. *TOY: A Multiparadigm Declarative System*. Proc. RTA'99, Springer LNCS 1631, pp 244–247, 1999.
- [20] F.J. López-Fraguas and J. Sánchez-Hernández. *A Proof Theoretic Approach to Failure in Functional Logic Programming*. Theory and Practice of Logic Programming 4(1), pp. 41–74, 2004.
- [21] J. Meseguer. *Conditional Rewriting Logic as a Unified Model of Concurrency*. Theoretical Computer Science 96, pp. 73–155, 1992.
- [22] T. Nipkow, L.C. Paulson, M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer LNCS 2283, 2002.
- [23] P. Padawitz. *Inductive Theorem Proving for Design Specifications*, J. Symbolic Computation 21, 41–99, 1996.
- [24] P. Padawitz. *Swinging Types = Functions + Relations + Transition Systems*, Theoretical Computer Science 243, 93–165, 2000.
- [25] M. Palomino Tarjuelo. *Comparing Meseguer's Rewriting Logic with the Logic CRWL*. Electronic Notes in Theoretical Computer Science 64, 22 pages, 2002.
- [26] L.C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*, Cambridge Univ. Press, 1987.
- [27] D. Pedreschi, S. Ruggieri. *Verification of Logic Programs*. J. Log. Program. 39 (1-3), pp. 125-176, 1999.
- [28] M. Rodríguez-Artalejo. *Functional and Constraint Logic Programming*. in H. Comon, C. Marché and R. Treinen (eds.), *Constraints in Computational Logics, Theory and Applications*, Revised Lectures of the International Summer School CCL'99, Springer LNCS 2002, Chapter 5, pp. 202–270, 2001.
- [29] Jaime Sánchez-Hernández, *Una Aproximación al fallo en programación declarativa multiparadigma*. PhD Univ. Complutense Madrid, 2004 (in spanish).
- [30] R.F. Stärk. *The theoretical foundations of LPTP (A logic program theorem prover)*. Journal of Logic Programming 36, pp. 241–269, 1998.