On T Logic Programming

Agostino Dovier*
Univ. di Verona, Ist. Policattedra, Fac. Scienze,
Strada Le Grazie 3, 37134 VERONA (I)
dovier@biotech.sci.univr.it

Andrea Formisano
Univ. di Roma *"La Sapienza"*, DSI,
Via Salaria 113, 00198 ROMA (I)
formisan@dsi.uniroma1.it

Alberto Policriti
Univ. di Udine, Dip. di Matematica e Informatica,
Via delle Scienze 206, 33100 UDINE (I)
policrit@dimi.uniud.it

Abstract

T-resolution parametrically generalizes standard resolution with respect to a first order theory T (the parameter). The inherent power of its derivation rule, however, makes it difficult to develop efficient unrestricted T-resolution based systems. $CLP(\mathcal{X})$ parametrically extends Horn clause logic programming with respect to a domain of computation $\mathcal{X}$. The theory T underlying the domain $\mathcal{X}$ is fixed *a-priori* and cannot be modified (extended) by the user.

In this paper we present the parametric logic programming language *T logic programming (TLP)* which extends the CLP-scheme by giving the possibility of acting on the theory T. The scheme is embedded into (linear) T-resolution; however, its syntax ensures that three rules (simple instances of the general T-resolution rule) are sufficient to implement the derivation process.

Keywords: Theorem Proving, Constraint Logic Programming, Foundations.

Introduction

T-resolution ([8]) parametrically generalizes standard resolution with respect to a first order theory T (the parameter). T-resolution was introduced to provide a general theorem-proving framework in which the power of the basic inference rule can grow with the underlying theory T. Another motivation was to establish criteria to classify first-order theories with respect to their suitability for theorem proving. In the framework obtained the parts of a proof which are carried out entirely within the theory are hidden inside

*The core of this work has been done when this author was affiliated at the Dip. di Matematica e Informatica of the Univ. of Udine, supported by C.N.R. grant No. 201.15.08.

the manager of the theory. Being binary, the T-resolution inference rule is simpler than the previously proposed *Theory-resolution* rule ([10]) and more suitable for integration with refinements already proposed of (classical) resolution. Moreover, the T-resolution rule ensures a complete independence between the background level (the T-decider) of the theorem prover and the foreground reasoner. This could not be the case for Theory-resolution, where the background reasoner plays an active role in the inference steps (*residues generation* [10]).

$CLP(\mathcal{X})$-scheme (cf., e.g., [5]) integrates the paradigms of *constraint solving* and *logic programming*. $CLP(\mathcal{X})$ parametrically extends Horn clause logic programming with respect to a specific domain of computation $\mathcal{X}$.

The theory T, on which an instance $CLP(\mathcal{X})$ of the CLP-scheme is based, is given *a-priori*. A privileged model $\mathcal{D}$ of T is chosen and model-theoretic, fixpoint, as well as operational semantics of the program are strongly based on $\mathcal{D}$. However, in CLP there is no way to affect the theory T by adding semantics to predicate and functional symbols. For instance, having a $CLP(\mathcal{R})$ system, one can compute in logic on a real domain. However, for an uninterpreted functional symbol f, there is no way to impose, for example, that $0 < f(X)$ for any X real.

In this paper, we present the parametric logic programming language, T-logic programming (TLP). TLP is an instance of the general T-resolution scheme and extends Constraint Logic Programming by providing a way for handling programs (consistently) extending T. In particular, we will provide inference rules instantiating the general T-resolution rule, realizing a simpler inference mechanism which behaves as the standard $CLP(T)$ one when the program is a CLP-program.

In CLP-schemata the only manner to share/exchange information between the theory and the program is through the use of the equality relator. In TLP, the theory T does not need to be an equational theory. Moreover, deduction steps are not bound to involve only pairs of literals, but the entire clause can be usefully employed.

Section 1 recalls basic definitions and results of T-resolution. Section 2 presents syntax and model-theoretic semantics of TLP language, and in Section 3 soundness and completeness results for the deduction system are proved. In Section 4 some further lines of research are drawn.

1 Preliminaries

Let T be a (consistent) first order theory whose axioms are in fully Skolemized form, Σ be a first order signature, and P a set of formulas. A formula φ is said *T-unsatisfiable* if for any model M of T it holds that $M \not\models \vec{\exists}\varphi$. We write: $T, P \vdash \varphi$ if and only if $P \cup \{\neg\varphi\}$ is T-unsatisfiable.

T-resolution is based on a binary rule

$$\frac{\alpha \quad \beta}{\gamma}$$

preserving T-satisfiability (i.e., $\alpha \wedge \beta$ is T-satisfiable implies γ is T-satisfiable). If T is the predicate calculus and α and β are clauses, then Robinson's resolution inference rule is exactly such rule (cf. Remark 1). T-resolution extends such methodology to any general theory T, provided a simple satisfiability test for T is available. A similar approach, *Theory resolution*, is presented in [10]; but in that case, an infinite number of rules is required by the author.

Given a first order theory T, two clauses $\alpha = \alpha_1 \vee \alpha_2$ and $\beta = \beta_1 \vee \beta_2$, and a substitution μ, the *T-resolution inference rule* can be described as follows:

$$\boxed{\frac{\alpha_1 \vee \alpha_2 \qquad \beta_1 \vee \beta_2}{(\alpha_1 \vee \beta_1)\mu} \quad T \vdash \vec{\forall}\Big(\big((\alpha_2 \wedge \beta_2) \rightarrow (\alpha_1 \vee \beta_1)\big)\mu\Big)}$$

More generally, γ is said to be *T-resolvent* of α and β if there exists a substitution μ such that one of the following conditions holds:

1. $\gamma = (\alpha_1 \vee \beta_1)\mu$ and $T \vdash \vec{\forall}((\alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1)\mu)$,
2. $\gamma = (\alpha \cup \{L\})\mu$ where either $L \in \beta$ or $\neg L \in \beta$.

Item (2) presents two cases: $L \in \beta$ is a particular case of (1) and is included for the sake of uniformity; $\neg L \in \beta$ is essential to guarantee the completeness of the rule (see [3]).

The rationale behind such rule can be understood assuming that $(\alpha_1 \vee \alpha_2) \wedge (\beta_1 \vee \beta_2)$ holds. The following cases are possible: if α_1 is true, then $\alpha_1 \vee \beta_1$ is true. Else, α_1 is false, hence α_2 must be true and we have two further cases: if β_1 is true, then again $\alpha_1 \vee \beta_1$ is true; otherwise, if β_2 is true, since it holds: $T \vdash (\alpha_2 \wedge \beta_2) \rightarrow (\alpha_1 \vee \beta_1)$, then $\alpha_1 \vee \beta_1$ is true.

Given a set P of clauses, a sequence $\gamma_1, \ldots, \gamma_n$ is a *derivation* of γ_n from P by T-resolution if every γ_i for $i \in \{1, \ldots, n\}$ is either in P or is a T-resolvent of γ_j and γ_k for $j, k < i$. γ_j and γ_k are said *parent-clauses* of γ_i. If $j = i - 1$, then the derivation is said *linear*; γ_j and γ_k are said *center-clause* and *side-clause*, respectively.

Remark 1 *Standard resolution is subsumed by T-resolution: if μ is a unifier of $\bar{s} = \bar{t}$, then it holds: $T \vdash \vec{\forall}((\neg p(\bar{s}) \wedge p(\bar{t}))\mu \rightarrow (\alpha_1 \vee \beta_1)\mu)$, since the l.h.s. formula of the implication is always false. This means that resolution inference rule:*

$$\frac{\alpha_1 \vee \neg p(\bar{s}) \qquad \beta_1 \vee p(\bar{t})}{(\alpha_1 \vee \beta_1)\mu}$$

is an instance of the T-resolution one.

Example 1 *Assume that T is a fragment of set theory dealing with $\in$ and $\cap$, and that $P = \{a \in b \cap c\}$. T-resolution allows to infer that $T, P \vdash a \in c$.*

$$\frac{a \in b \cap c \qquad a \notin c}{\square} \quad T \vdash a \in b \cap c \wedge a \notin c \rightarrow \texttt{false}$$

since $T \vdash a \in b \cap c \rightarrow a \in c$ holds.

T-resolution generalizes concepts as *semantic unification* already introduced for equational theories [9]: if T contains the standard equality axioms, and $T \vdash \vec{\forall}(\bar{s}\mu = \bar{t}\mu)$, then, as an instance of the general rule, we have:

$$\frac{\alpha_1 \vee \neg p(\bar{s}) \qquad \beta_1 \vee p(\bar{t})}{(\alpha_1 \vee \beta_1)\mu} \quad T \vdash \vec{\forall}(\bar{s}\mu = \bar{t}\mu).$$

Differently from other parametrical inference mechanisms (such as, e.g., CLP ([5])), T-resolution can be used for non-equational theories, as well.

It should be noticed that the power of the basic T-resolution rule allows to introduce trivial forms of linearity. As a matter of fact, given two clauses α and β, each of them can be obtained as T-resolvent of the pair: for instance, to obtain α as T-resolvent, let $\alpha_1 = \alpha$, $\alpha_2 = \texttt{false}$, $\beta_1 = \texttt{false}$, and $\beta_2 = \beta$; obviously, $\vec{\forall}(\alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1)$. This possibility allows to consider each derivation by T-resolution as "linear". This phenomenon destroys the main purpose of linear refinements and prevents any restriction/cut of the search space. We need a more restrictive concept of T-resolvent suitable to support non-trivial linearity.

In the context of linear derivations, a clause γ is said to be a *strict T-resolvent* of the center-clause α and the side-clause β if there exists a substitution μ such that one of the following conditions holds:

- $\gamma = (\alpha_1 \vee \beta_1)\mu$ and $T \vdash \vec{\forall}((\alpha_2 \wedge \beta_2 \rightarrow \alpha_1 \vee \beta_1)\mu)$, such that $(\alpha_1 \vee \beta_1)\mu$ does not include a variant of β;
- $\gamma = (\alpha \cup \{L\})\mu$ where either $L \in \beta$ or $\neg L \in \beta$.

Even when only strict T-resolvents are allowed, linear T-resolution is shown to be sound and complete, as well as other T-generalizations of standard resolution refinements ([3]). An implementation of linear T-resolution is employed in a theorem prover for polymodal logics ([2]).

Theorem 1 ([8]) *Let T be a first order theory. A set of clauses is T-unsatisfiable if and only if a clause having a T-unsatisfiable ground instance is derivable from it by linear T-resolution.*

The correctness and completeness result above does not depend from effectiveness of rule application. As for the CLP-scheme, a satisfiability test for simple formulas of T is needed (solvability, cf. [8]) to ensure mechanizability.

Linear T-resolution guarantees a clear distinction between the calculation level and the deduction level, and ensures the capability of integrating domain specific knowledge. These features are in strict analogy with the similar properties of CLP, born from the integration of Horn clause logic programming and (independently developed) constraint solvers, which represents the main reason of its practical usefulness.

2 T Logic Programming

In this section we introduce syntax and semantics of the deduction scheme, T Logic Programming (TLP). Operational semantics and a completeness theorem is given in the Section 3.

2.1 Syntax

Let $\Sigma = \Sigma_C \cup \Sigma_P$ be a first order signature. Σ_C is the *constraint* signature and Σ_P is the *program* signature. We impose: $\Sigma_C \cap \Sigma_P = \emptyset$, $\Sigma_C = \Pi_C \cup \mathcal{F}_C$, and $\Sigma_P = \Pi_P \cup \mathcal{F}_P$, where Π denotes a set of predicate symbols and $\mathcal{F}$ a set of functional (and constant) symbols. A Π-*atom* is an atom $p(t_1, \ldots, t_n)$, where $p \in \Pi$ and $t_1, \ldots, t_n$ are terms built from $\mathcal{F}_C \cup \mathcal{F}_P$ and a denumerable set of variables. A Π-*literal* is either a Π-atom or the negation of a Π-atom. There are cases in which T deals with all possible functional symbols (e.g., standard Clark Equality Theory); in such cases, clearly, $\mathcal{F}_P = \emptyset$.

The general form of a T-Logic Programming *program clause* is $B_0 \leftarrow B_1, \ldots, B_n$ where B_i can be either a Π_P-atom, or a Π_C-literal. If B_0 is a Π_C-literal, then $B_1, \ldots, B_n$ are Π_C-literals, as well. In this case the clause is said to be a *constraint clause*. A *goal* is a clause with empty head.

As opposed to predicate symbols of Π_P, predicates in Π_C can be (partially) defined in T which can be any first order theory. Moreover, they can occur in a negative literal either in the head or in the body of a TLP-program clause.

2.2 Model-theoretic semantics issues

The model-theoretic semantics of a definite program is based on the property of uniqueness of the least Herbrand model. Models of $CLP(\mathcal{X})$-programs, dealing with interpreted functional and predicate symbols over the domain $\mathcal{X}$, are not always Herbrand models. In general, $\mathcal{X}$ is provided with a first order theory T which can have several independent minimal models; a privileged model $\mathcal{D}_T$ of T is often chosen.

In CLP there is no way to acting on the theory T by modifying semantics of predicate and functional symbols of $\Sigma_C, \mathcal{F}_P$. For instance, having a $CLP(\mathcal{R})$ system, for two uninterpreted functional symbols $f, g \in \mathcal{F}_P$, we cannot impose that $g(X, X) < f(X)$ for any $0 < X < 1$.

The possibility of writing TLP-clauses with interpreted head allows to overcome such a restriction. There are three types of such program clauses:

1. clauses that add known (consistent) information to T. For instance, if T is a theory of natural numbers and the symbol '$<$' belongs to Π_C: $(0 < X) \leftarrow (1 < X)$.

2. Clauses which cause the inconsistency of $T \cup P$. With respect to the above theory T, for instance: $(x * x < 0) \leftarrow (0 < x)$ or, more implicitly, if T is a set theory with foundation axiom:
$$(a \in b) \leftarrow$$
$$(b \in a) \leftarrow \ .$$

3. Clauses which contain the functional symbols in $\mathcal{F}_P$ or $\mathcal{F}_C$ and consistently extend T, such as, for instance: $(f(X) < f(Y)) \leftarrow (X < Y)$.

We can characterize these three kind of clauses in model-theoretic terms: clauses of type 1) have, as models, all the models of T; none of the models of clauses of type 2) is model of T; only a proper subset of the models of T are models of clauses of kind 3).

We left to the programmer the task of avoiding situations of the type 2). Clauses of type 1) could be introduced in order to give a higher priority to some theorems of the theory T in the inference process.

Our aim would be to give a semantics to a program which extends a CLP-program by giving meaning to symbols in $\mathcal{F}_P$, or adding meaning to those in $\mathcal{F}_C$, when $T \cup P$ is consistent.

Let $\mathcal{D}_T$ be a model of T. In particular, it would be nice to find a privileged structure $\mathcal{D}$ which models $P \cup T$, such that the initial structure $\mathcal{D}_T$ is $\mathcal{D}$ restricted to Σ_C. Unfortunately, $P \cup T$ may extend T without guaranteeing the uniqueness of $\mathcal{D}$.

Since constraint clauses could contain negative Π_C-literals, typical semantical problems related to negation can arise: consider this example based on a set theory T:

$$(X \in a) \leftarrow (X \notin b)$$
$$(X \in b) \leftarrow (X \notin a).$$

The program is non-stratifiable ([1]) and at least two (minimal) independent models are possible: $[a \mapsto \{\emptyset\}, b \mapsto \emptyset]$ and $[a \mapsto \emptyset, b \mapsto \{\emptyset\}]$.

Even if stratification is assumed, there could be no unique model, consider, for example, the following two clauses

$$(0 \leq f(X)) \leftarrow$$
$$(f(X) \leq f(Y)) \leftarrow (X \leq Y)$$

stating that f is a monotone function whose co-domain is composed by elements greater than or equal to 0. There are infinitely many minimal interpretations for f.

In this paper we have not addressed the problem of providing suitable restrictions (for instance, of syntactical nature) on TLP-programs to ensure a precise semantical characterization in terms of uniqueness of the minimal model. In general, to force a unique minimal model of a TLP-program it is necessary that for each functional symbol $f \in \mathcal{F}_P$ enough knowledge is embedded in the program in order to guarantee/identify a unique interpretation.

However, soundness and completeness results presented in Section 3, are based directly on $P \cup T$ viewed as a first order theory, and hence they hold independently from the existence of a (minimal) unique model.

3 Operational Semantics

To mechanize TLP, we could use the T-resolution rule in its general form, as presented in Section 1. Completeness results relative to the general rule ensure that it can suitably simulate any CLP-scheme. In the following we introduce a restriction of the general linear T-resolution rule and we prove in Theorems 2 and 3 that it is suitable for our purposes.

Let μ be a substitution and $\leftarrow H_1, \ldots, H_k$ be a goal. We assume the body of a clause to be a multiset of literals. This guarantees that selecting H_1 will be as considering a generic element H_i of the multiset.

R1: If $B_0 \leftarrow B_1, \ldots, B_n$ is a TLP-clause, and B_0 and H_1 are Π_P-atoms,

$$\frac{B_0 \leftarrow B_1, \ldots, B_n \quad \leftarrow H_1, \ldots, H_k}{\leftarrow (B_1, \ldots, B_n, H_2, \ldots, H_k)\mu}$$
$$T \vdash \vec{\forall}(((B_0 \wedge \neg H_1) \rightarrow (\neg H_2 \vee \cdots \vee \neg H_k) \vee (\neg B_1 \vee \cdots \vee \neg B_n))\mu)$$

R2: If $\leftarrow B_1, \ldots, B_n$ is a constraint clause ($\neg B_i \leftarrow B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_n$) or a previously derived goal,

$$\frac{\leftarrow B_1, \ldots, B_n \quad \leftarrow H_1, \ldots, H_k}{\leftarrow (H_1, \ldots, H_r, B_1, \ldots, B_s)\mu}$$
$$T \vdash \vec{\forall}((\quad (\neg H_{r+1} \vee \cdots \vee \neg H_k) \wedge (\neg B_{s+1} \vee \cdots \vee \neg B_n) \rightarrow$$
$$(\neg H_1 \vee \cdots \vee \neg H_r) \vee (\neg B_1 \vee \cdots \vee \neg B_s))\mu)$$

where $H_{r+1}, \ldots, H_k, B_{s+1}, \ldots, B_n$ are all Π_C-literals. We discuss in more detail the features of this rule in the Section 4.

Loading: Let α be program clause, and L be a Π_C-literal,

$$\frac{\alpha \quad \leftarrow H_1, \ldots, H_k}{(\leftarrow H_1, \ldots, H_k, L)\mu} \quad L \text{ in } \alpha \text{ or } \neg L \text{ in } \alpha$$

Observe that strictness condition (cf. Section 1) holds for rule **R1**. Moreover, we impose that only strict applications of **R2** are allowed. This does not affect completeness and guarantees a proper linearity of the derivation.

Given a program P and a goal G, a *TLP-derivation* is a sequence of goals $G = G_0, \ldots, G_n$ such that, for each $i \in \{1, \ldots, n\}$, G_i is obtained from G_{i-1} and:

- a program clause using rule **R1** or **Loading**,
- a (constraint) program clause, or a goal G_k, $k < i$, using rule **R2**.

A *TLP-refutation* is a derivation of a goal $\leftarrow H_1, \ldots, H_k$, with H_i Π_C-literal (for all $i \in \{1, \ldots, k\}$), such that $H_1 \wedge \cdots \wedge H_k$ is T-satisfiable. For each $i \in \{1, \ldots, n\}$, let μ_i be the substitution employed obtaining G_i from G_{i-1}. The pair consisting of $\mu_1 \circ \cdots \circ \mu_n$ and $H_1 \wedge \cdots \wedge H_k$ is the *computed answer*.

In T-resolution it is not required to perform the satisfiability test for the constraint part (i.e., Π_C-literals) of each derived goal. Such a test was mandatory in the seminal definition of the CLP-scheme ([4]) and optional (apart for the last derived goal) in the more recent overview paper ([5]). However, in all implementations of T-resolution such a test is performed ([3, 7]) to improve efficiency and cut the search tree.

Observe that rule **R1** is sufficient to simulate the CLP derivation rule, provided the satisfiability checker is implemented.

Example 2 *Let T a theory over the reals dealing with $\leq$ and $\in$. Consider the simple program P defined by the following clauses:*

$$\begin{aligned} X \in a &\leftarrow 2 \leq X \\ p(X) &\leftarrow X \in a, X \leq 2 \end{aligned}$$

The first clause gives semantics to the constant symbol a in terms of $\in$, stating that it have to be interpreted as a set containing at least all numbers greater or equal to 2. The second clause defines a Π_P-literal (namely, $p(X)$) in terms of the new constraint $X \in a$. Submitting the goal: $\leftarrow p(X)$, few inference steps yield the goal $\leftarrow 2 \leq X, X \leq 2$.

Below we prove soundness and completeness results for the three rule introduced above:

Theorem 2 (Soundness) *Rules* **R1**, **R2**, *and* **Loading** *preserve T-satisfiability.*

Proof: It is immediate to see that the three rules are instances of the general T-resolution inference rule. Therefore, the soundness follows from Theorem 3.1 of [8]. □

Theorem 3 (Completeness) *Let P be a TLP-program, and G be a goal. If $P \cup \{G\}$ is a T-unsatisfiable set of clauses, then there exists a TLP-refutation of G.*

Proof: The proof will follow the classical pattern of proving ground completeness and then lifting the result to the general case.

To prove ground completeness we make use of the completeness of ground SL-T-resolution ([3]) (Subsumption Linear T-resolution, a refinement of linear T-resolution—cf. Section 1). Suppose that $P \cup \{G\}$ is a T-unsatisfiable set of ground clauses and that G is essential to T-unsatisfiability (i.e., P is T-satisfiable), then there exists a SL-T-refutation of $P \cup \{G\}$ with top-clause G: $G_0 = G, \ldots, G_n$. This SL-T-refutation has some useful properties: only loading, standard resolution, and the following instance of rule **R2**:

$$\frac{\alpha \quad \leftarrow H_1, \ldots, H_i, \ldots, H_k}{\leftarrow H_1, \ldots, H_{i-1}, H_{i+1}, \ldots, H_k} \quad T \vdash \vec{\forall}(H_1 \wedge \cdots \wedge H_{i-1} \wedge H_{i+1} \wedge \cdots \wedge H_k \rightarrow H_i)$$

called *unloading*, are used. Moreover, loading and unloading steps are performed on Π_C-literals only, and only standard resolution steps are performed on Π_P-literals. Notice that (viewing the program clauses and goals as disjunctions of literals) G consists of negative literals and that if a clause in P contains Π_P-literals, then exactly one of them is positive (the head of the clause). Those literals are the only positive Π_P-literals occurring in $P \cup \{G\}$. This ensures that each Π_P-literal occurring in any derived center-clause (i.e., G_i) is negative. To obtain a TLP-refutation from the given SL-T-refutation, the following steps are performed:

- standard resolution step on Π_P-literals: from the above argument, it follows that each of these steps must involve, as side-clause, a clause in P. Hence, these steps come out to be applications of rule **R1**;
- unloading and loading steps: these are particular cases of our TLP-rules (i.e., **R2** and **Loading**, respectively);
- standard resolution step on Π_C-literals: in this case we employ rule **R2** to perform exactly the same inference step. □

Notice that the derivation considered in the previous theorem are *strict*, in the sense introduced in Section 1.

Even though the proof of the above result is a direct consequence of the completeness of SL-T-resolution, the restrictions imposed on the T-resolution rule and on the kind of program clauses allowed in the context of TLP, make Theorem 3 rather significant. As a matter of fact, the two main differences between TLP and SL-T-resolution are the following: 1) TLP deals with sets of clauses built from two different (disjoint) sets of symbols (namely, Σ_C and Σ_P); 2) the part of the derivation relative to Π_P-literals is in fact a linear-input derivation. In other words, the inference process relative to Π_P-literals proceeds in a "Prolog-like fashion". Both these characteristics are not achievable in the general context of SL-T-resolution.

4 Conclusions and Further work

In this paper we have presented the scheme of logic programming language with constraints TLP. TLP extends the CLP-scheme, providing facilities to act on the theory T: interpreted symbols can occur in heads of clauses and (as a side effect) definition/characterization of entities is possible.

A deeper analysis of the model-theoretic semantics of TLP, also when the theory T is enlarged by using program defined predicate, is under investigation. As a matter of fact, it can be useful to give a semantics to new functional symbols and/or constants. For instance, it is possible to force the semantics of a constant (ω in the following example) in a such a way that in every model of the program, it is interpreted as the set of answers to certain goals ($\leftarrow num(X)$). As an example let T be a fragment of set theory, and consider the program P:

$$num(\emptyset) \leftarrow$$
$$num(X \cup \{X\}) \leftarrow num(X)$$
$$(X \in \omega) \leftarrow num(X)$$

The (minimal) semantics for the constant symbol ω is exactly the (infinite) set of all numerals *à la Von Neumann*.

A further step in this research will be to provide a (syntactical) characterization of classes of programs that, given a theory T, can be properly handled; moreover, it would be interesting to develop implementations that, for CLP-programs, have the same computational behavior as standard CLP interpreters.

The main source of potential inefficiency of a TLP-resolution procedure is the rule **R2**. This because, in general, previous derived goals have to be considered as possible side-clauses ($\leftarrow B_1, \ldots, B_n$). Therefore, the resulting TLP-derivation may not be an input-derivation. There are different possibilities to overcome this problem, each of them offers interesting research starting points. One approach involves the introduction of a stronger rule in place of **R2**, allowing as side-clauses only the clauses in P. This yields a characterization (by means of syntactical or semantical restrictions) of the class of those first order theories for which the completeness results still hold. Another possibility comes from considering an approach in the style of Model Elimination ([6]). In this case a suitable use, by means of particular literals inserted in the derived goals, of knowledge about the previous derivation steps, allows to preserve completeness even considering input derivations only.

Acknowledgements

We are grateful to the anonymous referees for their comments.

References

[1] Apt, K. R., and Bol, R. Logic Programming and Negation: a Survey. *Journal of Logic Programming 19,20* (1994), 9–71.

[2] van Benthem, J., D'Agostino, G., Montanari, A., and Policriti, A. Modal deduction in second-order logic and set theory - I. *Journal of Symbolic Computation 7*, 2 (1997). To appear.

[3] Formisano, A., and Policriti, A. T-resolution: Refinements and Model Elimination. Research Report 32/95, Dip. di Matematica e Informatica, Univ. di Udine, 1995. Available in ftp://ftp.dimi.uniud.it/pub/formisan/papers/TR32_12_95.ps.gz.

[4] Jaffar, J., and Lassez, J.-L. Constraint Logic Programming. Tech. rep., Department of Computer Science, Monash University, June 1986.

[5] Jaffar, J., and Maher, M. J. Constraint Logic Programming: A Survey. *The Journal of Logic Programming 19–20* (1994), 503–581.

[6] Loveland, D. W. *Automated Theorem Proving.* North-Holland, 1978.

[7] Masier, F. Ottimizzazione di algoritmi di soddisfacibilità mediante T-risoluzione. Master's thesis, Università di Udine, Dip. di Matematica e Informatica, 1996.

[8] Policriti, A., and Schwartz, J. T. T-theorem proving I. *Journal of Symbolic Computation 20*, 3 (1995), 315–342.

[9] Siekmann, J. H. Unification Theory. *Journal of Symbolic Computation 7* (1989).

[10] Stickel, M. E. Automated deduction by theory resolution. *Journal of Automated Reasoning I* (1985).