An Effective Bottom-Up Semantics for First-Order Linear Logic Programs

Marco Bozzano, Giorgio Delzanno, and Maurizio Martelli

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova
Via Dodecaneso 35, 16146 Genova - Italy
{bozzano,giorgio,martelli}@disi.unige.it

Abstract. We study the connection between algorithmic techniques for symbolic model checking [ACJT96,FS98,AJ99], and declarative and operational aspects of linear logic programming [And92,AP90]. Specifically, we show that the construction used to decide verification problems for Timed Petri Nets [AJ99] can be used to define a new fixpoint semantics for the fragment of linear logic called LO [AP90]. The fixpoint semantics is based on an effective T_P operator. As an alternative to traditional top-down approaches [And92,AP90,APC93], the effective fixpoint operator can be used to define a bottom-up evaluation procedure for first-order linear logic programs.

1 Introduction

Since its introduction in [Gir87], linear logic has been understood as a framework to reason about concurrent computations. Several researcher have in fact observed the existence of natural connections between linear logic and the theory of Petri Nets, see e.g., [Cer95,Kan94,MM91]. In this work we will investigate this connection focusing on the relations between algorithmic techniques used for the analysis of Petri Nets and provability in fragments of linear logic. The fragment we consider in this paper is called LO [AP90]. LO was originally introduced as a theoretical foundation for extensions of *logic programming* languages [And92]. As we will show next, LO programs enjoy a simple operational reading that makes clear the connection between provability in linear logic and verification methods in Petri Nets. Let us illustrate these ideas with the help of some examples.

Petri Nets in Propositional Linear Logic. Following [MM91], a Petri Net can be represented as a multiset-rewriting system over a finite alphabet $p, q, r, \ldots$ of *place* names. Among several possible ways, multiset rewrite rules can be expressed in linear logic using the connectives $\parr$ (multiplicative disjunction), and $\multimap$ (reversed linear implication). Multiplicative disjunction plays the role of multiset constructor, whereas linear implication can be used to define rewriting rules. Both connectives are allowed in the fragment LO. For instance, as shown in [Cer95] the LO clause

$$p \parr q \multimap p \parr p \parr q \parr t$$

can be viewed as a Petri Net *transition* that removes a token from places p and q and puts two tokens in place p, one in q, and one in t. Being a first-order language and thanks to the presence of other connectives, LO supports more sophisticated specifications than Petri Nets. For instance, Andreoli and Pareschi used LO clauses with occurrences of $\parr$ and & (additive conjunction) in their *body* to express what they called *external* and *internal* concurrency [AP90].

In our previous work [BDM00], we have defined an effective procedure to evaluate bottom-up LO propositional programs. Our construction is based on the *backward reachability* algorithm of [ACJT96] used to decide the so called *control state reachability problem* of Petri Nets (i.e. the problem of deciding if a given set of configurations are reachable from an initial one). The algorithm of [ACJT96] works as follows. Starting from a set of *target states*, the algorithm computes symbolically the transitive closure of the *predecessor* relation of the Petri Net taken into consideration (predecessor relation=transition relation read backwards). The algorithm is used to check safety properties: if the algorithm is executed starting from the set of *unsafe states*, the corresponding safety property holds if and only if the initial marking is not in the resulting fixpoint.

In order to illustrate the connection between backward reachability for Petri Nets and provability in LO, we first observe that LO clauses like

$$A_1 \parr \ldots \parr A_n \multimap \top$$

plays the same role that *facts* (unit clauses) do in logic programming. In fact, when applied in a resolution step, they generate instances of the LO axiom associated to the connective $\top$ (one of the logic constants of linear logic). Now, suppose we represent a Petri Net via an LO program P and the set of *target states* using a collection of LO facts T. Then, the set of *logical consequences* of the LO program $P \cup T$ will represent the set markings that are backward reachable from the target states. The algorithm we presented in [BDM00] is based on this idea, and it extends the backward reachability algorithm for Petri Nets of [ACJT96] to the more general case of propositional LO programs. This connection can be extended to more sophisticated classes of Petri Nets, as we discuss next.

Timed Petri Nets in First-Order Linear Logic. In Timed Petri Nets [AJ99], each token carries along its age. Furthermore, transitions are guarded by conditions over the age of tokens. To model the age of each token we can lift the specification in linear logic from the propositional to the first-order case. Basically, we can use an atomic formula $p(n)$ to model a token in place p with age n. For instance, the first-order linear logic rule (where for convenience, we use $\ldots \sqsubset c$ to denote a *constrained* formula in the style of Constraint Logic Programming [JM94])

$$p(X) \parr p(Y) \multimap r(Z) \sqsubset X \leq 1,\ Y \geq 2,\ Z \geq 1$$

can be read as a *discrete transition* of the Timed Petri Nets of [AJ99] that removes a token from place p if its *age* is less equal than 1, a token from place p

if and only if its *age* is greater or equal than 2, and adds a token with age greater or equal than 1 to place r. The previous clause can be seen as a compact way for expressing an infinite set of clauses (one for every evaluation of the variables which satisfies the constraints).

Recent results [ACJT96,AJ99] show that the *control state reachability problem*, under suitable hypothesis, is still decidable for Timed Petri Nets. The problem can be solved using the *backward reachability algorithm* of [ACJT96] in combination with a symbolic representation of sets of global states of Timed Petri Net via the so called *existential regions* [AJ99]. Now, can we exploit the results of [AJ99] for first-order (or even better, constrained) LO programs?

Our Contribution. In this paper we will show that it is possible to extend the parallel between the results on Petri Nets and provability in propositional LO, to new results relating Timed Petri Nets and provability in first-order LO. Specifically, we define a bottom-up fixpoint semantics for first-order LO programs (actually, we will consider the more general case of LO programs with constraints) using a *generalization* of the backward reachability algorithm for Timed Petri Nets of [AJ99]. Our procedure is a generalization in the following sense: we abstract away from the specific domain of Timed Petri Nets (e.g. where constraints are the *clock constraint* of [AJ99]); we handle the entire class of first-order LO programs (i.e. with nested occurrences of $\parr$ and & in the body of clauses). The resulting fixpoint semantics is based on an *effective* fixpoint operator and on a *symbolic* and *finite* representation of potentially infinite collections of first-order provable LO (atomic) goals. The termination of the fixpoint computation cannot be guaranteed in general, first-order LO programs are in fact Turing complete. As a consequence of the results in [AJ99,AN00], for a class of first-order LO programs that includes the *discrete component* of the Timed Petri Nets of [AJ99] the fixpoint semantics can be computed in finitely many steps.

Besides the relationships with Petri Nets, the new fixpoint semantics for first-order LO programs represents an interesting alternative to the traditional top-down semantics for linear logic programs studied in the literature [And92,AP90]. Our construction gives in fact an effective (though not complete) algorithm for the bottom-up evaluation of LO programs. Thus, also from the point-of-view of logic programming, we extend the applicability of our previous work [BDM00] (that was restricted to the propositional case) towards more interesting classes of linear logic programs.

Plan of the paper. After introducing some notations in Section 2, in Section 3 we present the language LO [AP90] enriched with constraints. In Section 4, we briefly review the approach we followed in [BDM00], and we extend it to the first-order case. In Section 5 we discuss the connection between LO fixpoint semantics and the reachability algorithm for Timed Petri Nets presented in [AJ99]. Finally, we discuss related work and conclusions in Sections 6 and 7. The proofs of all results are in the extended version of the paper [BDM00a].

2 Preliminaries and Notation

In this paper we will consider first-order linear logic languages built upon a signature Σ comprising a finite set of term constructors, a finite set of predicate symbols, and a denumerable set of variable symbols. We will denote term constructors by $a, b, \ldots, f, g, \ldots$, predicate symbols by $p, q, r, \ldots$, and variables by $X, Y, Z, \ldots$. The set of (possibly non ground) atoms over Σ will be denoted A_Σ. We will use $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ to denote multisets of (possibly non ground) atoms. We denote a multiset $\mathcal{A}$ with (possibly duplicated) elements $A_1, \ldots, A_n$ by $\{A_1, \ldots, A_n\}$ or simply $A_1, \ldots, A_n$ if this notation is not ambiguous. A multiset $\mathcal{A}$ is uniquely determined by a finite map Occ from A_Σ to the set of natural numbers, such that $Occ_{\mathcal{A}}(A)$ is the number of occurrences of A in $\mathcal{A}$. The *multiset inclusion* relation $\preccurlyeq$ is defined as follows: $\mathcal{A} \preccurlyeq \mathcal{B}$ iff $Occ_{\mathcal{A}}(A) \leq Occ_{\mathcal{B}}(A)$ for every A. The *empty* multiset is denoted ϵ and is such that $Occ_{\epsilon}(A) = 0$ for every A (clearly, $\epsilon \preccurlyeq \mathcal{A}$ for any $\mathcal{A}$). The *multiset union* $\mathcal{A}, \mathcal{B}$ (written $\mathcal{A} + \mathcal{B}$ when ',' is ambiguous) of $\mathcal{A}$ and $\mathcal{B}$ is such that $Occ_{\mathcal{A},\mathcal{B}}(A) = Occ_{\mathcal{A}}(A) + Occ_{\mathcal{B}}(A)$ for every A. The *multiset difference* $\mathcal{A} \setminus \mathcal{B}$ is such that $Occ_{\mathcal{A}\setminus\mathcal{B}}(A) = max(0, Occ_{\mathcal{A}}(A) - Occ_{\mathcal{B}}(A))$ for every A. Finally, we define an operation $\bullet$ to compute the *least upper bound* of two multisets with respect to $\preccurlyeq$. Namely, $\mathcal{A} \bullet \mathcal{B}$ is such that $Occ_{\mathcal{A}\bullet\mathcal{B}}(A) = max(Occ_{\mathcal{A}}(A), Occ_{\mathcal{B}}(A))$ for every A.

In the rest of the paper we will use $\Delta, \Theta, \ldots$ to denote multisets of possibly compound formulas. Given two multisets Δ and Θ, $\Delta \preccurlyeq \Theta$ indicates multiset inclusion and Δ, Θ multiset union, as before, and $\Delta, \{G\}$ is written simply Δ, G. In the following, we will refer to a multiset of goal-formulas as a *context*. Given a linear disjunction of atomic formulas $H = A_1 \parr \ldots \parr A_n$, we also introduce the notation $\widehat{H}$ to denote the multiset $A_1, \ldots, A_n$.

3 The Language LO Enriched with Constraints

LO [AP90] is a logic programming language based on a fragment of linear logic defined over the linear connectives $\multimap$, $\&$, $\parr$, and $\top$. In this paper we will present a first-order formulation of LO using *constraints*. We will use constraints as a means to *represent concisely collections of ground program clauses* defined over a parametric interpretation domain.

Constraints. Let $\mathcal{V}$ be a denumerable set of variables. In this paper a *constraint* is a conjunction (in the 'classical' sense) of atomic predicates $c_1 \wedge \ldots \wedge c_n$, where c_i has variables from $\mathcal{V}$. The interpretation domain $\mathcal{D}$ of the constraints is fixed a priori. As an example, linear arithmetic constraints are conjunctions of inequalities of the form $k_1 X_1 + \ldots + k_n X_n$ *op* k, where k_i is an integer constant, X_i is a variable that ranges over integer (real) numbers for all $i : 1, \ldots, n$, *op* is one operator taken from $\leq, \geq, <, >, =$, and k is an integer constant (e.g., $2X + 3X \geq 1 \wedge X \geq 0 \wedge Y \geq 0$). An *evaluation* is an assignment σ which maps variables in $\mathcal{V}$ to values in $\mathcal{D}$. We denote the result of applying an evaluation σ to a constraint c by $\sigma(c)$. A *solution* for a constraint c is an evaluation σ such

that $\sigma(c)$ is true. Note that values assigned to variables which do not appear in c can be disregarded. For instance, any evaluation which maps X to 0.5, Y to 2 and Z to 1.7 is a solution for the constraint (over the real numbers) $X \leq 1 \wedge Y \geq 2 \wedge Z \geq 1$. We call $Sol(c)$ the set of solutions of c. In this paper we will always consider a constraint language with equality, so that $t_1 = t_2$ is always defined for any expression t_1, t_2. This property ensures that it is always possible to express *unification constraints*. Given two constraints c_1 and c_2, we define

$$c_1 \textbf{ entails } c_2 \text{ if and only if } Sol(c_1) \subseteq Sol(c_2)$$

(e.g. $X = 2$ **entails** $X \geq 2$). In this paper we will limit ourselves to consider domains in which the relation **entails** is *decidable*, and there are two constraints $true$ and $false$ such that c **entails** $true$, and $false$ **entails** c for any c.

Multisets Unifiers. Let $\mathcal{A} = p_1(\mathbf{x_1}), \ldots, p_n(\mathbf{x_n})$ and $\mathcal{B} = q_1(\mathbf{y_1}), \ldots, q_n(\mathbf{y_n})$ be two multisets of atomic formulas such that $\mathbf{x_i}$ and $\mathbf{y_i}$ are vectors of variables (distinct from each other) for $i : 1, \ldots, n$. If $p_i = q_i$ (and they have the same arity) for $i : 1, \ldots, n$, then the two multisets are *unifiable*. The resulting unifier will be the constraint $\mathbf{x_1} = \mathbf{y_1} \wedge \ldots \wedge \mathbf{x_n} = \mathbf{y_n}$. Since the order of atoms inside a multiset does not count, there can be more than one way for two multisets to be unified. In the following, we will use the notation $\mathcal{A} = \mathcal{B}$ to denote one constraint which is *non-deterministically* selected from the set of unifiers of $\mathcal{A}$ and $\mathcal{B}$. If there are no permutations of one of the two multisets such that the previous conditions are satisfied, then $\mathcal{A} = \mathcal{B}$ will denote the constraint $false$. Finally, in case $n = 0$ we have the constraint $\epsilon = \epsilon$ which stands for $true$.

LO Programs. We first define a *goal* formula via the following grammar.

$$G \ ::= \ G \ \parr \ G \ \mid \ G \ \& \ G \ \mid \ p(\mathbf{x}) \ \mid \ \top.$$

Here $\mathbf{x}$ is a vector of variables from the set $\mathcal{V}$. An LO program with constraints consists of a set of universally quantified formulas having the following form:

$$A_1 \parr \ldots \parr A_n \ \circ\!\!- \ G \ \Box \ c,$$

where $n \geq 1$, $A_i = p_i(\mathbf{x_i})$ for $i : 1, \ldots, n$, and $\mathbf{x_i}$ is a vector of variables in $\mathcal{V}$, G is a goal formula, and c is a constraint whose *scope* extends over the whole implication $A_1 \parr \ldots \parr A_n \ \circ\!\!- \ G$. Note that we assume that all compound terms occur in c. For instance, using constraints we would write the first-order LO clause $p(f(X)) \circ\!\!- q(X)$ as $p(Y) \circ\!\!- q(X) \ \Box \ Y = f(X)$. Given a constrained expression $F \ \Box \ c$ (F can be a clause, a goal, a multiset of atoms, etc.) we define

$$Gnd(F \ \Box \ c) = \{\sigma(F) \mid \sigma \in Sol(c)\}.$$

We say that $A \in Gnd(F \ \Box \ c)$ is an *instance* of $F \ \Box \ c$. The definition can be extended to sets of expressions in the canonical way. Thus, given a program P, $Gnd(P)$ denotes the set of all instances of clauses in P. Furthermore, note that a G-formula by itself is not very interesting, whereas *constrained goals* like $p(X) \ \& \ q(Y) \ \Box \ X - Y \geq 1$ are the counterpart of the goals of Constraint Logic Programming [JM94].

$$\frac{}{P \Rightarrow \top, \Delta}\ \top_r \qquad \frac{P \Rightarrow G_1, G_2, \Delta}{P \Rightarrow G_1 \parr G_2, \Delta}\ \parr_r \qquad \frac{P \Rightarrow G_1, \Delta \quad P \Rightarrow G_2, \Delta}{P \Rightarrow G_1 \,\&\, G_2, \Delta}\ \&_r$$

$$\frac{P \Rightarrow G, \mathcal{A}}{P \Rightarrow \widehat{H}, \mathcal{A}}\ bc \ \text{ provided } H \multimap G \in Gnd(P)$$

Fig. 1. Proof system for LO.

Example 1. Let $C = p(X,Y) \parr q(Z) \multimap t(Z,Y) \Box X = f(Z)$. If we interpret constraints over the term algebra $\{a, f(a), \ldots\}$, then $p(f(a),a) \parr q(a) \multimap t(a,a) \in Gnd(C)$.

Top-down Ground Semantics. We define the *top-down* operational semantics of LO with constraints using the *uniform* (goal-driven) proof system of [AP90], presented in Fig. 1. As said at the beginning of this section, we introduce constraints just as a convenient means to represent sets of ground clauses, therefore the proof system of [AP90] is sufficient for our purposes. In Fig.1, P is a set of implicational clauses, $\mathcal{A}$ denotes a multiset of atomic formulas, whereas Δ denotes a multiset of G-formulas. A sequent is provable if all branches of its proof tree terminate with instances of the $\top_r$ axiom. The proof system of Fig. 1 is a specialization of more general uniform proof systems for linear logic like Andreoli's focusing proofs [And92], and Forum [Mil96]. The rule *bc* denotes a backchaining (resolution) step. Note that *bc* can be executed only if the right-hand side of the current LO sequent consists of atomic formulas. Thus, LO clauses behave like *multiset* rewriting rules. LO clauses having the form $H \multimap \top \Box c$, where c is a satisfiable constraint, play the same role as the unit clauses of Horn programs. In fact, a backchaining step over these clauses *succeeds* independently of the current context. This observation shows that the *weakening* rule is *admissible* in LO, i.e., if $P \Rightarrow \Delta$ is provable, then $P \Rightarrow \Delta'$ is provable for any $\Delta \preccurlyeq \Delta'$. Finally, the *success set* or *operational semantics* of a (ground) LO program P is defined as

$$O(P) = \{\mathcal{A} \mid \mathcal{A} \textit{ is a ground multiset and } P \Rightarrow \mathcal{A} \textit{ is provable}\}.$$

Remark 1 (Interpretation of constraints and $\Box$ in Linear Logic.). At least in principle, it seems possible to model constrained LO clauses inside linear logic itself. For this purpose, however, we need fragments larger than LO. In presence of the connective $\otimes$, the constrained LO formula

$$H \multimap G \Box c_1 \wedge \ldots \wedge c_n$$

could be represented as the linear logic formula

$$H \multimap (G \otimes (c_1 \otimes \ldots \otimes c_n)),$$

while provability of constraint formulas could be expressed using a specialized (linear logic) theory.

Bottom-Up Ground Semantics. In [BDM00], we have defined a ground fixpoint semantics based on a new fixpoint operator T_P for LO programs. We recall here the main ideas (see [BDM00] for the details). Given a finite alphabet of propositional atoms (symbols), the (ground) Herbrand base B_P is defined as

$$B_P = \{\mathcal{A} \mid \mathcal{A} \text{ is a ground multiset of atoms in P}\}.$$

We say that $I \subseteq B_P$ is a Herbrand interpretation. Herbrand interpretations form a complete lattice wrt set inclusion. Satisfiability of a context (i.e. a multiset of goals) Δ in a given interpretation I, is defined via the judgment $I \models \Delta[\mathcal{A}]$. Let us assume that I is a set of provable multisets. Then, the *output* $\mathcal{A}$ of the judgment $I \models \Delta[\mathcal{A}]$ is any multiset of *resources* such that $\Delta + \mathcal{A}$ is provable.

Definition 1 (Ground Satisfiability). *Let $I \subseteq B_P$, then $\models$ is defined as follows:*

$I \models \top, \mathcal{A}[\mathcal{A}']$ *for any ground multiset* $\mathcal{A}'$;

$I \models \mathcal{A}[\mathcal{A}']$ *if* $\mathcal{A} + \mathcal{A}' \in I$;

$I \models G_1 \parr G_2, \Delta[\mathcal{A}]$ *if* $I \models G_1, G_2, \Delta[\mathcal{A}]$;

$I \models G_1 \,\&\, G_2, \Delta[\mathcal{A}]$ *if* $I \models G_1, \Delta[\mathcal{A}]$ *and* $I \models G_2, \Delta[\mathcal{A}]$.

Given a program P, the operator T_P for constrained LO programs is defined as follows:

$$T_P(I) = \{\widehat{H} + \mathcal{A} \mid H \multimap G \in \mathit{Gnd}(P),\ I \models G[\mathcal{A}]\}.$$

The fixpoint semantics, defined as the least fixpoint of T_P, is sound and complete with respect to the operational semantics, as stated in the following theorem.

Theorem 1 ([BDM00]). *For every LO program P, $O(P) = \mathit{lfp}(T_P)$.*

4 Towards an Effective Non-ground Semantics

The bottom-up ground semantics for first-order LO is not effective for two different reasons: there might be possibly infinite instantiations of a constrained LO clause (this makes the condition $H \multimap G \in \mathit{Gnd}(P)$ not effective); there are infinitely many *output* contexts in a satisfiability judgment (this makes the computation of $I \models \Delta\ [\mathcal{A}]$ not effective). In [BDM00], we have shown how to circumvent the second problem in the propositional case.

Propositional Case. By exploiting the fact that *weakening* is admissible in LO, in [BDM00] we noted that a provable multiset $\mathcal{A}$ can be used to implicitly represent the set of provable multisets $\mathit{Up}(\mathcal{A}) = \{\mathcal{B} \mid \mathcal{A} \preccurlyeq \mathcal{B}\}$ where $\preccurlyeq$ is multiset inclusion, i.e., $\mathit{Up}(\mathcal{A})$ is the *ideal* generated by $\mathcal{A}$ wrt $\preccurlyeq$. As in the ground semantics, interpretations are still collections of multisets, however the denotation of an interpretation I becomes now $[\![I]\!] = \bigcup_{\mathcal{A} \in I} \mathit{Up}(\mathcal{A})$. Since multiset inclusion is a well-quasi ordering [ACJT96], the termination of the fixpoint computation is guaranteed by choosing the following *pointwise* ordering of interpretations:

$$I \sqsubseteq J \text{ iff } \forall\ \mathcal{A} \in I\ \exists\ \mathcal{B} \in J\ \mathcal{B} \preccurlyeq \mathcal{A}.$$

Note that, $\mathcal{A} \preccurlyeq \mathcal{B}$ implies $Up(\mathcal{B}) \subseteq Up(\mathcal{A})$, whereas $I \sqsubseteq J$ implies $[\![I]\!] \subseteq [\![J]\!]$. The property of being a well-quasi order can be lifted from $\preccurlyeq$ to $\sqsubseteq$. Based on this observation, it is possible to define an *effective* operator whose fixpoint is computable in finitely many steps (see [BDM00] for its definition). In the following section we will lift these ideas to first-order programs.

First-Order Case. In view of our main goal (the definition of an effective fixpoint semantics for first order LO), we first define a new notion of Herbrand interpretation based on the notion of *constrained multiset*. Constrained multisets will be our *symbolic representation* of sets of ground multisets. A constrained multiset $\mathcal{A} \ \Box \ c$ is a multiset of atomic formulas $\mathcal{A}$ whose free variables are constrained by c, e.g., like $p(X), q(Y) \ \Box \ X \geq Y$ (Note that the whole multiset is in the scope of the constraint). Variables which appear in c but not in $\mathcal{A}$ are *implicitly* considered *existentially* quantified, i.e. for instance $p(X), q(Y) \ \Box \ X \geq Y \ \wedge \ Y = Z$ is logically equivalent to $p(X), q(Y) \ \Box \ \exists Z.(X \geq Y \wedge Y = Z)$. In the following. we will use $\mathbf{M}, \mathbf{N}, \ldots$ to denote constrained multisets. Now, we extend the ideas used in the propositional case as follows. A constrained multiset $\mathbf{M}$ defined as $\mathcal{A} \ \Box \ c$ will represent the collection of provable goals that satisfy *at least* the constraint (wrt the multiset component $\mathcal{A}$ and the constraint part c) imposed by $\mathbf{M}$. Formally,

$$[\![\mathcal{A} \ \Box \ c]\!] = Up(Gnd(\mathcal{A} \ \Box \ c)),$$

the denotations are defined by taking first all instances of $\mathcal{A} \ \Box \ c$, and then (as in the propositional case) taking their upward closure wrt $\preccurlyeq$. As an example, the denotation (over the real numbers) of $\mathbf{M} = (p(X), q(Y) \Box X \geq Y)$ contains both $\{p(1), q(0)\}$ and $\{p(1), q(0), q(9)\}$, since they both satisfy *at least* the constraint in $\mathbf{M}$. We are now ready to define the new notion of interpretation.

Definition 2 (Extended Herbrand Base and Interpretations). *The extended base is defined as* $B_P = \{ \ \mathcal{A} \ \Box \ c \mid \mathcal{A}$ *is a multiset of atoms* $p(\mathbf{x})$, c *is a constraint* $\}$. $I \subseteq B_P$ *is called extended Herbrand interpretation and*

$$[\![I]\!] = \bigcup_{(\mathcal{A} \ \Box \ c) \in I} [\![\mathcal{A} \ \Box \ c]\!].$$

Definition 3 (Lattice of Interpretations). *The lattice* $\langle \mathcal{I}, \sqsubseteq \rangle$ *of extended Herbrand interpretations is defined as follows:*

- $\mathcal{I} = \mathcal{P}(B_P)/\simeq$ *where* $I \simeq J$ *if and only if* $[\![I]\!] = [\![J]\!]$*;*
- $[I]_\simeq \sqsubseteq [J]_\simeq$ *if and only if* $[\![I]\!] \subseteq [\![J]\!]$*;*
- *the bottom element is the empty set* $\emptyset$*, the top element is the* $\simeq$*-equivalence class of the singleton* $\{(\epsilon \ \Box \ true)\}$*;*
- *the least upper bound* $I \sqcup J$ *is the* $\simeq$*-equivalence class of* $I \cup J$*.*

The equivalence $\simeq$ allows us to reason modulo *redundancies*. For the sake of simplicity, in the rest of the paper we will identify an interpretation I with its class $[I]_\simeq$. In the following, we will lift the fixpoint semantics from the propositional to the first-order case using the new notion of interpretation. To this aim, we will first need an effective notion of satisfiability wrt an interpretation.

4.1 Effective Satisfiability Test

To decide if a given constrained goal $G \sqsupset c$ is satisfiable in the extended interpretation I, we must extend the judgments used in the propositional case in order to handle the constraint components. Since G may contain nested occurrences of connectives, it might be necessary to decompose G into a multiset of atoms $\mathcal{A}$ and then match $\mathcal{A}$ against I. The last step may introduce new *bindings* for the variables in G. We must return these bindings in form of constraints, since they will be used when incorporated in the backward application of a clause $H \multimap G$ (e.g., whenever H and G share common variables). For this purpose, we introduce the new judgment

$$I \Vdash \mathbf{G}\ [\mathcal{C}\ ;\ c]$$

where $\mathbf{G}$ is a constrained goal (later extended to a constrained multiset of goals), $\mathcal{C}$ is the output multiset (as in the propositional case) and c is the output constraint. The judgment $I \Vdash \mathbf{G}\ [\mathcal{C}\ ;\ c]$ should return in $\mathcal{C}$ a representation of all possible multisets which, added to $\mathbf{G}$, make it provable (under the constraint c). Based on the previous intuition, we will formally define the satisfiability relation as follows. In the following the notation $G, \Delta \sqsupset c$ must be read as $(G, \Delta) \sqsupset c$.

Definition 4 (Non-ground Satisfiability Relation $\Vdash$). *Let $I \in \mathcal{I}$, then $\Vdash$ is defined as follows:*

ALL $\quad I \Vdash \top, \mathcal{A} \sqsupset c\ [\epsilon\ ;\ c]$;

PAR $\quad I \Vdash G_1 \parr G_2, \Delta \sqsupset c\ [\mathcal{C}\ ;\ c']$ *if and only if* $I \Vdash G_1, G_2, \Delta \sqsupset c\ [\mathcal{C}\ ;\ c']$;

MULT $\quad I \Vdash \mathcal{A} \sqsupset c\ [\mathcal{C}\ ;\ c']$ *if and only if there exists a renamed copy* $\mathcal{B} \sqsupset d$ *of an element of* I, $\mathcal{B}' \preccurlyeq \mathcal{B}$, *and* $\mathcal{A}' \preccurlyeq \mathcal{A}$ *such that* $c' \equiv \mathcal{B}' = \mathcal{A}' \wedge c \wedge d$ *is satisfiable, and* $\mathcal{C} = \mathcal{B} \setminus \mathcal{B}'$;

WITH $\quad I \Vdash G_1 \,\&\, G_2, \Delta \sqsupset c\ [\ \mathcal{C}\ ;\ c']$ *if and only if* $I \Vdash G_1, \Delta \sqsupset c\ [\ \mathcal{C}_1\ ;\ c_1]$, $I \Vdash G_2, \Delta \sqsupset c\ [\ \mathcal{C}_2\ ;\ c_2]$, *and there exist* $\mathcal{C}_1' \preccurlyeq \mathcal{C}_1$, $\mathcal{C}_2' \preccurlyeq \mathcal{C}_2$, *such that* $c' \equiv \mathcal{C}_1' = \mathcal{C}_2' \wedge c_1 \wedge c_2$ *is satisfiable, and* $\mathcal{C} = \mathcal{C}_1 + (\mathcal{C}_2 \setminus \mathcal{C}_2')$.

The above definition is entirely algorithmic. Given an interpretation and a constrained goal G (a multiset of goals, in general), the set $\{\langle \mathcal{C}, c\rangle \mid I \Vdash \mathbf{G}\ [\mathcal{C}\ ;\ c]\}$ is always finite. Rules PAR and ALL should be clear. In rule WITH, given the output context $\mathcal{C}_1 \sqsupset c_1$ and $\mathcal{C}_2 \sqsupset c_2$ for the two conjuncts, the output context for $G_1 \,\&\, G_2$ is obtained by merging in all possible ways sub-multisets of $\mathcal{C}_1$ and $\mathcal{C}_2$ that are unifiable. Rule MULT is for goals consisting of a constrained multiset $\mathcal{A} \sqsupset c$ of atomic formulas. If $\mathcal{A}$ contains a sub-multiset unifiable with a sub-multiset $\mathcal{B}'$ of an element $\mathcal{B}$ of I, $\mathcal{B} \setminus \mathcal{B}'$ is the minimal context to be added to $\mathcal{A}$.

Example 2. Consider a language over the term universe $\{a, f(a), \ldots\}$, let $\mathbf{G}$ be the goal $(q(X) \parr r(Y))\ \&\ s(Z) \sqsupset Z = a$, and I be the interpretation consisting of the two constrained multisets

$$\mathbf{M}_1 \equiv t(U_1, V_1), q(W_1) \sqsupset U_1 = f(Q_1) \wedge W_1 = a,$$
$$\mathbf{M}_2 \equiv t(U_2, V_2), s(W_2) \sqsupset V_2 = f(Q_2) \wedge W_2 = a.$$

Using the WITH rule, we have to compute $\mathcal{C}_1, \mathcal{C}_2, c_1, c_2$ such that $I \Vdash q(X) \parr r(Y) \sqsubset Z = a\ [\mathcal{C}_1\ ;\ c_1]$ and $I \Vdash s(Z) \sqsubset Z = a\ [\mathcal{C}_2\ ;\ c_2]$. For the first conjunct and using the PAR rule, we have that $I \Vdash q(X) \parr r(Y) \sqsubset Z = a\ [\mathcal{C}_1\ ;\ c_1]$ iff $I \Vdash q(X), r(Y) \sqsubset Z = a\ [\mathcal{C}_1\ ;\ c_1]$. By MULT rule, applied to $\mathbf{M}_1 \in I$, we have that $\mathcal{C}_1 = t(U_1, V_1)$ and $c_1 \equiv X = W_1 \wedge W_1 = a \wedge Z = a \wedge U_1 = f(Q_1)$. For the second conjunct and using the MULT rule applied to $\mathbf{M}_2 \in I$, we have that $\mathcal{C}_2 = t(U_2, V_2)$ and $c_2 \equiv Z = W_2 \wedge Z = a \wedge W_2 = a \wedge V_2 = f(Q_2)$. Therefore by definition of the WITH rule, if we unify $t(U_1, V_1)$ and $t(U_2, V_2)$, we have that

$$I \Vdash \mathbf{G}\ [t(U_1, V_1)\ ;\ U_1 = U_2 \wedge V_1 = V_2 \wedge c_1 \wedge c_2].$$

More concisely, by renaming the variables, we get $I \Vdash \mathbf{G}\ [t(U, V)\ ;\ X = a \wedge Z = a \wedge U = f(Q) \wedge V = f(Q')]$. We also have, by choosing empty sub-multisets in WITH rule, that

$$I \Vdash \mathbf{G}\ [t(U_1, V_1), t(U_2, V_2)\ ;\ c_1 \wedge c_2].$$

More concisely, by renaming the variables, we get $I \Vdash \mathbf{G}\ [t(U, V), t(U', V')\ ;\ X = a \wedge Z = a \wedge U = f(Q) \wedge V' = f(Q')]$.

4.2 Effective Fixpoint Semantics

Based on the previous definitions, we can define a fully symbolic fixpoint operator S_P for LO programs with constraints. The operator S_P transforms extended interpretations in extended interpretations as follows.

Definition 5 (Fixpoint Operator S_P). *Given an LO program P, and $I \in \mathcal{I}$,*

$$S_P(I) = \{\ \widehat{H} + \mathcal{C} \sqsubset c')\ |\ \exists\ (H \circ\!\!- G \sqsubset c)\ \textit{variant of a clause in } P \textit{ s.t. } I \Vdash\ G \sqsubset c\ [\mathcal{C}\ ;\ c']\ \}$$

Proposition 1. *S_P is monotonic and continuous over the lattice $\langle \mathcal{I}, \sqsubseteq \rangle$.*

Then, we have the following results.

Proposition 2. *Let P be a constrained LO program, and $I \in \mathcal{I}$. Then, $[\![S_P(I)]\!] = T_{Gnd(P)}([\![I]\!])$.*

Corollary 1. *$[\![lfp(S_P)]\!] = lfp(T_{Gnd(P)})$.*

Let $SymbF(P) = lfp(S_P)$. Then, we have the following main Theorem.

Theorem 2 (Soundness and Completeness). *Given a constrained LO program P, $O_{Gnd(P)} = [\![SymbF(P)]\!]$.*

Though a single application of the operator S_P is effective, in general it might be impossible to compute the fixpoint of S_P (first-order LO programs are Turing complete). We can make a similar observation for the non-ground operator S_P of Constraint Logic Programming [JM94].

Example 3. Consider the clause $p(X) \parr p(Z) \circ\!\!- (q(X) \parr r(Y)) \ \& \ s(Z) \Box Z = a$, and the interpretation I of Example 2. Let $\mathbf{G} = (q(X) \parr r(Y)) \ \& \ s(Z) \Box Z = a$. From Example 2, we know that $I \Vdash \mathbf{G}\ [t(U,V)\ ;\ X = a \wedge Z = a \wedge U = f(Q) \wedge V = f(Q')]$ and $I \Vdash G\ [t(U,V), t(U',V')\ ;\ X = a \wedge Z = a \wedge U = f(Q) \wedge V' = f(Q')]$. Thus, we get that $S_P(I)$ contains

$$p(X), p(Z), t(U,V) \Box X = a \wedge Z = a \wedge U = f(Q) \wedge V = f(Q'),$$
$$p(X), p(Z), t(U,V), t(U',V') \Box X = a \wedge Z = a \wedge U = f(Q) \wedge V' = f(Q').$$

As particular instances, we have that $S_P(I)$ includes (representations of) the multisets $p(a), p(a), t(f(a), f(a))$ and $p(a), p(a), t(f(a), a), t(a, f(a))$. This example shows the importance of maintaining all constraints generated during an evaluation of a judgment (e.g. $X = a$ in the first case). In fact, the scope of these constraints extends over the atoms in the *head* of clauses (e.g. $p(X)$).

5 Relationship with Timed Petri Nets

We will illustrate the main ideas of the connection between our fixpoint semantics and the framework of [AJ99] to decide reachability problems for Timed Petri Nets. As explained in the introduction, Timed Petri Nets (TPNs) are infinite-state networks where tokens move from place to place (place=node of the net) carrying along their age. TPN transitions have two possible forms. Discrete transitions specify how tokens move around the network, whereas timed transitions simply increase the age of every token in the network. The results of [AJ99] are based on a *symbolic* representation of potentially infinite sets of TPN configurations (i.e. tokens with their age) via *existential regions*. An existential region is a formula having the following form

$$\exists x_1 \ldots \exists x_n.\ P(x_1, ..., x_n) \ \textit{and}\ c(x_1, ..., x_n)$$

whose meaning is as follows: *there exist* at least n distinct *tokens distributed in the network as described by formula P and whose ages satisfy the constraint c.* The formula P is a conjunction of constraints of the form $x_j \in p_i$, whose meaning is *the token x_j is in place p_i*. In the interpretation domain existentially quantified variables are required to denote *distinct* values. The constraint c is a *clock constraint* [AJ99], i.e., a conjunction of atomic predicates like $x_i - x_j \leq k$ (where k is an integer constant), which expresses a bound on the difference between the clock values of different tokens. More simple constraints like $x_i \leq k$ or $x_i \geq k$, which limit the clock value of a single token, can be seen as subcases of the previous class of constraints. An existential region Φ denotes an *upward closed* set of TPN configurations (i.e. those satisfying *at least* the constraints in Φ). Based on this construction, the algorithm of [AJ99] solves the following type of reachability problems.

***Control state reachability*:** given an initial configuration I_0 and an existential region Φ, is it possible to *reach* a configuration in (the denotation of) Φ starting from (an instance of) I_0?

The algorithm works as follows. It computes the effect of applying *backwards* the transitions of the TPN starting from Φ, until it reaches a fixpoint F. As a last step, it checks if $[\![I_0]\!] \cap [\![F]\!] = \emptyset$. The termination of the algorithm is guaranteed by the following properties: existential regions are *well-quasi ordered* [AJ99,AN00]; the class of existential regions is closed under backward applications of TPN transitions, i.e., if we apply backwards TPN transitions to a set of existential regions we still obtain a set of existential regions. Instead of entering in more details in the TPN formalism, we immediately show how the discrete components of TPNs can be modeled via LO programs.

TPNs as First-Order Linear Logic Theories. First of all, a token in place p and age n can be represented as the atomic predicate $p(n)$. Thus, discrete transitions can be represented via LO clauses of the form:

$$p_1(X_1) \parr \ldots \parr p_n(X_n) \multimap q_1(Y_1) \parr \ldots \parr q_m(Y_m) \Box c$$

where p_i, q_j are place names and X_i, Y_j are variables associated to the ages, and c is a constraint over the age of tokens. Specifically, c specifies the constraint under which the tokens can be removed from $p_1, \ldots, p_n$, and the new constraint on the ages of the tokens added to $q_1, \ldots, q_m$. Timed transitions cannot be represented directly in LO. This encoding can be done via a meta-rule

$$\frac{P \Rightarrow p_1(X_1 + \delta), \ldots, p_n(X_n + \delta)}{P \Rightarrow p_1(X_1), \ldots, p_n(X_n)} \; Time \;\; (\delta \geq 0)$$

whose definition in Linear Logic (e.g. in Forum [Mil96]) requires giving a *family* of clauses, using the constant $\mathbf{1}$ to ensure that the age of *every* token in the network is incremented as a result of a timed transition. In contrast with $\top$, $\mathbf{1}$ succeeds only in the empty context. The semantics for LO presented here could be extended, similarly to what done in the propositional case [BDM00], to include the constant $\mathbf{1}$. Anyway, as far as the application to TPNs is concerned, in [AJ99] the authors show how to compute effectively the backward application of timed transitions (i.e. our meta-rule) on a given existential region. We skip therefore a detailed discussion about encoding timed transitions in LO. We can use the ideas of [AJ99] provided we find a counterpart of existential regions in the LO setting, and provided we can find a way to connect a *backward reachability step* with the LO operational semantics.

Existential Regions as Constrained Multisets. Existential regions can be naturally represented as *constrained* multisets having the following form:

$$p_1(X_1), \ldots, p_n(X_n) \Box c$$

where c is a constraint. In fact, the denotation $[\![A \Box c]\!]$ of a constrained multiset $\mathcal{A} \Box c$, captures precisely the intended meaning of the existential regions of [AJ99]. As an example, $[\![p(X), q(Y) \Box 0 \leq X - Y \leq 1]\!] = \{\{p(1), q(0)\}, \{p(1), q(0), q(3.6)\}, \ldots\}$.

Backward Reachability = Bottom-Up Evaluation. At this point, it should be clear that computing the effect of a backward application of the transitions of a TPN N coincides with computing an application of the S_{P_N} operator associated to the LO program P_N encoding N. In order to *start up* the bottom-up evaluation from the set of configurations represented by the target existential region Φ, we simply have to add the clause: $p_1(X_1) \parr \ldots \parr p_m(X_m) \multimap \top \Box c$ where $\Phi = \exists x_1, \ldots x_m.\ x_1 \in p_1$ *and* $\ldots$ *and* $x_m \in p_m$ *and* c. Through this encoding, we automatically inherit from [AJ99] the following property.

Theorem 3. *Let N be a TPN, Φ an existential region, and P_N be their encoding in LO enriched with the meta-rule for timed transitions. Then, $SymbF(P_N)$ is computable in finitely many steps.*

Besides the basic multiset rewriting mechanism, the linear logic language LO provides other connectives that add further expressiveness to the operational reading of specifications. For instance, we give some hints about using the additive conjunction & to specify an operation to hierarchically compose TPNs. If we think about TPNs as representing the execution of some kind of *protocol*, the rule $p(X) \multimap p_1(X_1) \ \& \ p_2(X_2) \Box c$ could be seen as the specification that the protocol p will start (at the time specified by c) two sub-protocols that must succeed in order for p to succeed. The subsystems p_1 and p_2 will run independently. Since clock constraints are closed under conjunctions, the result of Theorem 3 can be extended in order to include LO programs with conjunction as the one in the previous example.

6 Related Works

To our knowledge, our work is the first attempt to connect algorithmic techniques used in symbolic model checking with declarative and operational aspects of linear logic programming. In [BDM00], we have considered the relation between *propositional* LO and Petri Nets. In this paper we have extended the connection to first-order LO programs and more general notions of Petri Nets.

In [HW98], Harland and Winikoff present an abstract deductive system for bottom-up evaluation of linear logic programs. The left introduction plus weakening and cut rules are used to compute the logical consequences of a given formula. Though the framework is given for a more general fragment than LO, it does not provide for an *effective* procedure to evaluate programs.

Finally, in [Cer95] Cervesato shows how to encode Petri Nets in LO, Lolli and Forum exploiting the different features of these languages; whereas in [APC93], Andreoli, Pareschi and Castagnetti define an improved *top-down* strategy for *propositional* LO based on the Karp-Miller's coverability tree of Petri Nets, i.e., a *forward* exploration with accelerations.

7 Conclusions and Future Work

In this paper we have investigated the connections between techniques used for symbolic model checking of infinite-state systems [ACJT96,AJ99,FS98] and

provability for first-order linear logic programs [AP90]. We have generalized the construction used in [AJ99] to decide verification problems for Timed Petri Nets in order to build an effective *fixpoint* semantics for first-order LO programs. Ad hoc notions like the existential regions of [AJ99] find a natural counterpart as elements of the non-ground interpretations of LO programs, a notion inspired by the more advanced semantics of Constraint Logic Programming [GDL95,JM94]. Furthermore, we have shown that the algorithms used for (Timed) Petri Nets can be extended in order to capture the richer specification language LO.

The main interest of the new semantics is that it gives us a way to evaluate bottom-up first-order LO programs, i.e., an alternative operational semantics that could be useful to study new applications of linear logic programming (as discussed in [HW98]). For this reason, we think it would be important to extend our method to other linear logic languages like Lolli [HM94] and Forum [Mil96].

The work presented in this paper can also be a source of further investigations concerning the analysis of programs and the development of new observable semantics. In particular, the semantics could be extended in order to cope with observables like the *non ground success set* and *computed answer substitutions* [FLMP93] of LO programs. To this aim, we plan to formulate the constraint-based semantics presented here using a more traditional approach based on substitutions and most general unifiers. While the constraint-based formulation was particularly suitable to study the connection with TPNs, a formulation based on substitutions could be useful for extending traditional program analysis techniques to linear logic programs.

Acknowledgments. The authors would like to thank Iliano Cervesato and the anonymous reviewers for their useful comments and suggestions.

References

[ACJT96] P. A. Abdulla, K. Cerāns, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. of LICS 96*, pages 313–321, 1996.

[AJ99] P. A. Abdulla, and B. Jonsson. Ensuring Completeness of Symbolic Verification Methods for Infinite-State Systems. To appear in *Theoretical Computer Science*, 1999.

[AN00] P. A. Abdulla, and A. Nylen. Better is Better than Well: On Efficient Verification of Infinite-State Systems. In *Proc. LICS'2000*, pages 132-140, 2000.

[And92] J. M. Andreoli. Logic Programming with Focusing proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[AP90] J. M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-In Inheritance. In *Proc. of ICLP'90*, pages 495–510, 1990.

[APC93] J. M. Andreoli, R. Pareschi and T. Castagnetti. Abstract Interpretation of Linear Logic Programming. In *Proc. of ILPS'93*, pages 295–314, 1993.

[BDM00] M. Bozzano, G. Delzanno, and M. Martelli. A Bottom-up semantics for Linear Logic Programs. In *Proc. of PPDP 2000*, pages 92-102, 2000.

[BDM00a] M. Bozzano, G. Delzanno, and M. Martelli. An Effective Bottom-Up Semantics for First-Order Linear Logic Programs and its Relationship with Decidability Results for Timed Petri Nets. Technical Report, DISI, Università di Genova, Decemeber 2000.

[Cer95] I. Cervesato. Petri Nets and Linear Logic: a Case Study for Logic Programming. In *Proc. of GULP-PRODE'95*, pages 313–318, 1995.

[DP99] G. Delzanno and A. Podelski. Model Checking in CLP. In *Proc. of TACAS'99*, LNCS 1579, pages 223–239, 1999.

[FLMP93] M. Falaschi and G. Levi and M. Martelli and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 102(1):86-113, 1993.

[FS98] A. Finkel and P. Schnoebelen. Well-structured Transition Systems Everywhere! Technical Report LSV-98-4, Laboratoire Spécification et Vérification, ENS Cachan, 1998. To appear in *Theoretical Computer Science*, 1999.

[Fri00] L. Fribourg. Constraint logic programming applied to model checking. In *Proc. LOPSTR'99*, pages 31-42, 2000.

[GDL95] M. Gabbrielli, M. G. Dore and G. Levi. Observable Semantics for Constraint Logic Programs. *Journal of Logic and Computation*, 5(2): 133–171, 1995.

[Gir87] J. Y. Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987.

[HM94] J. S. Hodas and D. Miller. Logic Programming in a Fragment of Intuitionistic Linear Logic. *Information and Computation*, 110(2):327–365, 1994.

[HW98] J. Harland and M. Winikoff. Making Logic Programs Reactive. In *Proc. of JICSLP'98 Workshop Dynamics'98*, pages 43–58, 1998.

[JM94] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[Kan94] M. I. Kanovich. Petri Nets, Horn Programs, Linear Logic, and Vector Games. In *Proc TACS. '94*, pages 642–666, 1994.

[KMM+97] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *Proc. CAV '97*, pages 424–435, 1997.

[Kop95] A. P. Kopylov. Propositional Linear Logic with Weakening is Decidable. In *Proc. LICS 95*, pages 496-504, 1995.

[Llo87] J. W. Lloyd. Foundations of Logic Programming. Springer-Verlag, 1987.

[MM91] N. Martí-Oliet, J. Meseguer. From Petri Nets to Linear Logic. Mathematical Structures in Computer Science 1(1):69–101, 1991.

[Mil96] D. Miller. Forum: A Multiple-Conclusion Specification Logic. *Theoretical Computer Science*, 165(1):201–232, 1996.