A Bottom-up Semantics for Linear Logic Programs

Marco Bozzano[*] Giorgio Delzanno Maurizio Martelli[†]

Dipartimento di Informatica e
Scienze dell'Informazione
Via Dodecaneso 35
16146 Genova, Italy
giorgio@disi.unige.it

ABSTRACT

The operational semantics of linear logic programming languages is given in terms of goal-driven sequent calculi. The proof-theoretic presentation is the natural counterpart of the *top-down* semantics of traditional logic programs. In this paper we investigate the theoretical foundation of an alternative operational semantics based on a *bottom-up* evaluation of linear logic programs via a fixpoint operator. The bottom-up fixpoint semantics is important in applications like active databases, and, more in general, for program analysis and verification. As a first case-study, we consider Andreoli and Pareschi's LO [4] enriched with the constant **1**. For this language, we give a fixpoint semantics based on an operator defined in the style of T_P. We give an algorithm to compute a single application of the fixpoint operator where constraints are used to represent in a finite way possibly infinite sets of provable goals. Furthermore, we show that the fixpoint semantics for propositional LO without the constant **1** can be computed after finitely many steps. To our knowledge, this is the first attempt to define an effective fixpoint semantics for LO. We hope that our work will open interesting perspectives for the analysis and verification of linear logic programming languages.

1. INTRODUCTION

In recent years a number of fragments of linear logic [16] have been proposed as a logical foundation for extensions of logic programming [28]. Several new programming languages like LO [4], ACL [24], Lolli [20], and Lygon [17] have been proposed with the aim of enriching traditional logic programming languages like Prolog with a well-founded notion of state and with aspects of concurrency. The operational semantics of this class of languages is given via a sequent-calculi presentation of the corresponding fragment of linear logic. Special classes of proofs like the focusing proofs of Andreoli [2] and the uniform proofs of Miller [27] allow us to restrict our attention to *cut-free*, *goal-driven* proof systems that are complete with respect to provability in linear logic. These presentations of linear logic are the natural counterpart of the traditional *top-down* operational semantics of logic programs.

[*]email:bozzano@disi.unige.it

[†]email:martelli@disi.unige.it

In this paper we investigate an alternative operational semantics for one of the above mentioned languages, namely LO [4]. The operational semantics we propose consists of a *goal-independent bottom-up* evaluation of LO programs. Specifically, given an LO program P our aim is to compute a finite representation of the set of goals that are provable from P. There are several reasons to look at this problem. First of all, as discussed in [18], the bottom-up evaluation of programs is the key ingredient for all applications where it is difficult or impossible to specify a given goal in advance. Examples are active (constraint) databases, agent-based systems and genetic algorithms. Furthermore, new results connecting verification techniques and semantics of logic programs [12] show that bottom-up evaluation can be used to automatically check *properties* (specified in temporal logic like CTL) of the original program. Finally, a formal definition of the bottom-up semantics can be useful for studying equivalence, compositionality and abstract interpretation (as for traditional logic programs [7, 15]). The reason we selected LO in this preliminary work is that we were looking for a relatively simple linear logic language with a uniform-proof presentation, state-based computations and aspects of concurrency. Operationally, LO programs behave like a set of *multiset* rewriting rules. In practice, LO has been successfully applied to model, e.g., concurrent object-oriented languages [4], and coordination languages based on the Linda model [3].

Technically, our contributions are as follows. We first consider a formulation of LO with $⅋, \multimap, \&$ and $\top$. Following the semantic framework of (constraint) logic programming [15, 21], we formulate the bottom-up evaluation procedure in two steps. We first define what one could call a *ground* semantics via a *fixpoint* operator T_P defined over an extended notion of Herbrand interpretation consisting of *multisets* of atomic formulas. This way, we capture the *uniformity* of LO-provability, according to which compound goals must be completely decomposed into atomic goals before program clauses can be applied. Due to the notion of *resource* peculiar to linear logic, this semantics may introduce infinitely many provable multisets even for propositional goals. In fact, LO-provability enjoys the following property. If a multiset of goals Δ is provable in P, then any Δ' such that Δ is a sub-multiset of Δ' is provable in P. To circumvent this problem, we order the interpretations according to the multiset inclusion relation of their elements and we define a new operator S_P that computes only the *minimal* (wrt. multiset inclusion) provable contexts. Dickson's Lemma (see e.g. [1]) ensures the termination of the fixpoint computation based on S_P for propositional LO programs. Interestingly, this result is an instance of the general decidability results for model checking of infinite-state systems given in [1, 14]. In the paper we also show that adding the constant $\mathbf{1}$ to the original formulation of LO in [4] breaks down the above mentioned property. Nevertheless, it is still possible to define an effective $S_P^{\mathbf{1}}$ operator by taking *linear constraints* as *symbolic* representation of potentially infinite sets of contexts (actually, the previous result is a particular case where constraints have no equalities). Though for the new operator we cannot guarantee that the fixpoint can be reached after finitely many steps, this connection allows us to apply techniques developed in model checking for infinite-state systems (e.g. [1, 9, 12, 19, 14]) and abstract interpretation [11] to compute approximations of the fixpoint of $S_P^{\mathbf{1}}$. In this paper we limit ourselves to the study of the propositional case that, as shown in [6], can be viewed as the target of a possible abstract interpretation of a first-order program. To our knowledge, this is the first attempt of defining an effective fixpoint semantics for LO with the $\mathbf{1}$ constant. We hope that this work will help in finding new research directions (e.g. connections with model checking) and application for linear logic programs.

Plan of the paper.

After introducing some notations in Section 2, in Section 3 we recall the main feature of LO [4]. In Section 4 we introduce the *ground* semantics, the operator T_P and prove that the least fixpoint of T_P characterizes the operational semantics of a program. In Section 5 we introduce the *symbolic* S_P operator and we relate it to T_P. In Section 6 we consider an extended fragment with $\mathbf{1}$, extending the notion of satisfiability given in Section 4 and introducing an operator $T_P^{\mathbf{1}}$. In Section 7 we introduce a symbolic operator $S_P^{\mathbf{1}}$ for the extended fragment, and we discuss its algorithmic implementation in Section 8. Finally, in Section 9 and Section 10 we discuss related works and conclusions.

An extended version of this paper (containing all the proofs) is available as technical report [8].

2. PRELIMINARIES

In the paper we will use $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ to denote multisets of propositional symbols, hereafter called *facts*, defined over a fixed signature $\Sigma = \{a_1, \ldots, a_n\}$. A fact $\mathcal{A}$ is uniquely determined by a finite map $Occ : \Sigma \rightsquigarrow \mathrm{N}$ such that $Occ_{\mathcal{A}}(a_i)$ is the number of occurrences of a_i in $\mathcal{A}$. Facts are ordered according to the *multiset inclusion* relation $\preccurlyeq$ defined as follows, $\mathcal{A} \preccurlyeq \mathcal{B}$ if and only if $Occ_{\mathcal{A}}(a_i) \leq Occ_{\mathcal{B}}(a_i)$ for $i : 1, \ldots, n$. The *empty* multiset is denoted ϵ and is such that $Occ_{\epsilon}(a_i) = 0$ for $i : 1, \ldots, n$, and $\epsilon \preccurlyeq \mathcal{A}$ for any $\mathcal{A}$. The *multiset union* $\mathcal{A}, \mathcal{B}$ (alternatively $\mathcal{A} + \mathcal{B}$ when ',' is ambiguous) of two facts $\mathcal{A}$ and $\mathcal{B}$ is such that $Occ_{\mathcal{A},\mathcal{B}}(a_i) = Occ_{\mathcal{A}}(a_i) + Occ_{\mathcal{B}}(a_i)$ for $i : 1, \ldots, n$. The *multiset difference* $\mathcal{A} \setminus \mathcal{B}$ is such that $Occ_{\mathcal{A}\setminus\mathcal{B}}(a_i) = max(0, Occ_{\mathcal{A}}(a_i) - Occ_{\mathcal{B}}(a_i))$ for $i : 1, \ldots, n$. Finally, we define a special operation $\bullet$ to compute the *least upper bound* of two facts with respect to $\preccurlyeq$. Namely, $\mathcal{A} \bullet \mathcal{B}$ is such that $Occ_{\mathcal{A}\bullet\mathcal{B}}(a_i) = max(Occ_{\mathcal{A}}(a_i), Occ_{\mathcal{B}}(a_i))$ for $i : 1, \ldots, n$. 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, a *context* will denote a multiset of goal-formulas (a *fact* is a context in which every formula is atomic). Given a linear disjunction of atomic formulas $H = a_1 \invamp \ldots \invamp a_n$, we introduce the notation $\widehat{H}$ to denote the multiset $a_1, \ldots, a_n$. Finally, let $T : \mathcal{I} \rightsquigarrow \mathcal{I}$ be an operator defined over a complete lattice $\langle \mathcal{I}, \sqsubseteq \rangle$. We define $T{\uparrow}_0 = \emptyset$, where $\emptyset$ is the bottom element, $T{\uparrow}_{k+1} = T(T{\uparrow}_k)$ for all $k \geq 0$, and $T{\uparrow}_\omega = \bigsqcup_{k=0}^{\infty} T{\uparrow}_k$, where $\bigsqcup$ is the least upper bound wrt. $\sqsubseteq$. Furthermore, we use $\mathit{lfp}(T)$ to denote the *least fixpoint* of T.

3. THE PROGRAMMING LANGUAGE LO

LO [4] is a logic programming language based on linear logic. Its mathematical foundations lie on a proof-theoretical presentation of a fragment of linear logic defined over the linear connectives $\multimapinv$, $\&$, $\invamp$, and $\top$. In the propositional case LO consists of the following class of formulas:

$$\mathbf{D} ::= \mathbf{A}_1 \invamp \ldots \invamp \mathbf{A}_n \multimapinv \mathbf{G} \mid \mathbf{D} \,\&\, \mathbf{D}$$
$$\mathbf{G} ::= \mathbf{G} \invamp \mathbf{G} \mid \mathbf{G} \,\&\, \mathbf{G} \mid \mathbf{A} \mid \top$$

Here $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and $\mathbf{A}$ range over propositional symbols from a fixed signature Σ. $\mathbf{G}$-formulas correspond to *goals* to be evaluated in a given program. $\mathbf{D}$-formulas correspond to multiple-headed *program clauses*. An LO program is a $\mathbf{D}$-formula. Let P be the program $C_1 \,\&\, \ldots \,\&\, C_n$. The execution of a multiset of $\mathbf{G}$-formulas $G_1, \ldots, G_k$ in P corresponds to a goal-driven proof for the two-sided LO-sequent

$$P \Rightarrow G_1, \ldots, G_k.$$

The LO-sequent $P \Rightarrow G_1, \ldots, G_k$ is an abbreviation for the following two-sided linear logic sequent

$$!C_1 \otimes \ldots \otimes !C_n \rightarrow G_1 \invamp \ldots \invamp G_k.$$

The formula $!F$ on the left-hand side of a sequent indicates that F can be used in a proof an arbitrary number of times. From the left rules of $\otimes$ and $\&$ (see e.g. [2, 27]) this implies that an LO-Program can be viewed also as a *set of reusable clauses*. According to this view, the operational semantics of LO is given via the *uniform* (goal-driven) proof system defined in Fig. 1. In Fig.1, P is a set of implicational clauses, $\mathcal{A}$ denotes a multiset of atomic formulas, whereas Δ denotes a multiset of $\mathbf{G}$-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 [2], and Forum [27]. The rule *bc* denotes a backchaining (resolution) step. Note that *bc*

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

$$\frac{P \Rightarrow G, \mathcal{A}}{P \Rightarrow a_1, \ldots, a_n, \mathcal{A}}\ bc \ \ (a_1 ⅋ \ \ldots ⅋ \ a_n \circ\!- G \ \in \ P)$$

Figure 1: A proof system for LO

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 following form

$$a_1 ⅋ \ \ldots ⅋ \ a_n \circ\!- \top$$

play the same role as the unit clauses of Horn programs. In fact, a backchaining step over such a clause leads to *success* independently of the current context $\mathcal{A}$, as shown in the following scheme:

$$\frac{\dfrac{}{P \Rightarrow \top, \mathcal{A}}\ \top_r}{P \Rightarrow a_1, \ldots, a_n, \mathcal{A}}\ bc \ \ (a_1 ⅋ \ \ldots ⅋ \ a_n \circ\!- \top \in P)$$

This observation leads us to the following property.

Proposition 3.1. *Given an LO program P and two contexts Δ, Δ' such that $\Delta \preccurlyeq \Delta'$, if $P \Rightarrow \Delta$ then $P \Rightarrow \Delta'$.*

This property is the key point in our analysis of the operational behavior of LO. It states that the *weakening* rule is *admissible* in LO.

Example 3.2. Let P be the LO program consisting of the clauses

1. $a \circ\!- (b \,\&\, c) ⅋ e$
2. $b \circ\!- \top$
3. $c \circ\!- d ⅋ b$

and consider an initial goal a. Using clause 1., to prove a we have to prove $(b \,\&\, c) ⅋ e$, and then, by LO $⅋_r$ and $\&_r$ rules, we have to prove b, e and c, e. By clause 2., b, e is provable, and by clause 3. to prove c, e we can prove d, b, e, which is in turn provable by clause 2. By Proposition 3.1, provability of a implies provability of any multiset of goals containing a. □

4. A BOTTOM-UP SEMANTICS FOR LO

The proof-theoretical semantics of LO corresponds to the *top-down* operational semantics based on resolution for traditional logic programming languages like Prolog. The main difference is that instead of conjunctions of atomic formulas (as in Prolog) in LO we need to handle arbitrary nesting of conjunctions, expressed via $\&$, and disjunctions, expressed via $⅋$, of goals. In this paper we are interested in finding a suitable definition of *bottom-up* semantics that can be used as an alternative operational semantics for LO. More precisely, given an LO program P we would like to compute all goal formulas G such that G is provable in P. Without loss of generality, we will limit ourselves to goal formulas consisting of multisets of atomic formulas. In fact, in order to analyze a compound goal G, we can always add a clause $p \circ\!- G$ to the original program and analyze p. For this purpose, we define the *operational* semantics of an LO program P as follows:

$$O(P) = \{\mathcal{A} \mid \mathcal{A} \ is \ a \ fact \ and \ P \Rightarrow \mathcal{A} \ is \ provable\}$$

In the rest of the paper we will always consider propositional LO programs defined over a *finite* set of propositional symbols Σ. We give the following definitions.

Definition 4.1 (Herbrand base B_P). *Given a propositional LO program P defined over Σ,*

$$B_P = \{\mathcal{A} \mid \mathcal{A} \textit{ is a multiset (fact) over } \Sigma\}$$

Definition 4.2 (Herbrand interpretation). *We say that $I \subseteq B_P$ is a Herbrand interpretation. Herbrand interpretations form a complete lattice $D = \langle \mathcal{P}(B_P), \subseteq \rangle$ with respect to set inclusion.*

Before introducing the formal definition of the *ground* bottom-up semantics, we need to define a notion of satisfiability of a context Δ in a given interpretation I. For this purpose, we introduce the judgment $I \models \Delta[\mathcal{A}]$. In $I \models \Delta[\mathcal{A}]$, the *output* $\mathcal{A}$ is a fact such that $\mathcal{A}+\Delta$ is valid in I. This notion of *satisfiability* is modeled according to the right-introduction rules of the connectives. The notion of output fact $\mathcal{A}$ will simplify the presentation of the algorithmic version of the judgment which we will present in Section 5.

Definition 4.3 (Satisfiability). *Let $I \subseteq B_P$, then $\models$ is defined as follows:*

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

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

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

The relation $\models$ satisfies the following properties.

Lemma 4.4. *For any interpretations I, J, context Δ, and fact $\mathcal{A}$,*

1. *$I \models \Delta[\mathcal{A}]$ if and only if $I \models \Delta, \mathcal{A}[\epsilon]$;*

2. *if $I \subseteq J$ and $I \models \Delta[\mathcal{A}]$ then $J \models \Delta[\mathcal{A}]$;*
3. *given a chain of interpretations $I_1 \subseteq I_2 \subseteq \dots$, if $\bigcup_{i=1}^{\infty} I_i \models \Delta[\mathcal{A}]$ then there exists k s.t. $I_k \models \Delta[\mathcal{A}]$.*

We now come to the definition of the fixpoint operator T_P.

DEFINITION 4.5 (FIXPOINT OPERATOR T_P). *Given a program P, the operator T_P is defined as follows:*

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

The following property holds.

PROPOSITION 4.6. *For every program P, T_p is monotonic and continuous wrt. $\subseteq$.*

Monotonicity and continuity of the T_P operator imply, by Tarski Theorem, that $lfp(T_P) = T_P{\uparrow}_\omega$.

Following [26], we define the *fixpoint semantics* $F(P)$ of an LO program P as the least fixpoint of T_P, namely $F(P) = lfp(T_P)$. Intuitively, $T_P(I)$ is the set of *immediate logical consequences* of the program P and of the facts in I. In fact, if we define P_I as the program $\{A \multimapinv \top \mid \widehat{A} \in I\}$, the definition of T_P can be viewed as the following instance of the *cut* rule of linear logic:

$$\frac{!P, G \rightarrow H \quad !P_I \rightarrow G, \mathcal{A}}{!P, !P_I \rightarrow H, \mathcal{A}}\ cut$$

Using the notation used for LO-sequents we obtain the following rule:

$$\frac{P \Rightarrow H \multimapinv G \quad P_I \Rightarrow G, \mathcal{A}}{P \cup P_I \Rightarrow H, \mathcal{A}}\ cut$$

Note that, since $H \multimapinv G \in P$, the sequent $P \Rightarrow H \multimapinv G$ is always provable in linear logic. According to this view, $F(P)$ characterizes the set of *logical consequences* of a program P.

The fixpoint semantics is sound and complete with respect to the operational semantics as stated in the following theorem.

THEOREM 4.7 (SOUNDNESS AND COMPLETENESS). *For every LO program P, $F(P) = O(P)$.*

We note that it is possible to define a *model-theoretic* semantics, based on the classical notion of *least model* with respect to a given class of models and partial order relation. In this context, the partial order relation is simply set inclusion, while models are exactly Herbrand interpretations which satisfy program clauses, i.e. I is a model of P if and only if for every clause $H \multimapinv G \in P$ and for every fact $\mathcal{A}$,

$$I \models G[\mathcal{A}] \text{ implies } I \models H[\mathcal{A}].$$

It turns out that the operational, fixpoint and model-theoretic semantics are all equivalent. We note that these semantics are also equivalent to the *phase semantics* for LO given in [4].

5. AN EFFECTIVE SEMANTICS FOR LO

The operator T_P defined in the previous section is not *effective*. As an example, take the program P consisting of the clause $a \multimapinv \top$. Then, $T_P(\emptyset)$ contains all multisets with *at least* one occurrence of a. In other words, $T_P(\emptyset) = \{\mathcal{B} \mid a \preccurlyeq \mathcal{B}\}$, where $\preccurlyeq$ is the multiset inclusion relation of Section 2. In order to compute *effectively* one step of T_P, we have to find a *finite* representation of potentially infinite sets of facts (in the terminology of [1], a *constraint system*). The previous example suggests us that a provable fact $\mathcal{A}$ may be used to *implicitly* represent the ideal generated by $\mathcal{A}$, i.e., the subset of B_P defined as follows:

$$[\![\mathcal{A}]\!] = \{\mathcal{B} \mid \mathcal{A} \preccurlyeq \mathcal{B}\}$$

We extend the definition of $[\![\cdot]\!]$ to sets of facts as follows: $[\![I]\!] = \bigcup_{\mathcal{A} \in I} [\![\mathcal{A}]\!]$. Based on this idea, we define an *abstract* Herbrand base where we handle every single fact $\mathcal{A}$ as a representative element for $[\![\mathcal{A}]\!]$ (note that in the semantics of Section 4 the denotation of a fact $\mathcal{A}$ is $\mathcal{A}$ itself!). The abstract domain is defined as follows.

DEFINITION 5.1 (ABSTRACT INTERPRETATION). *The lattice $\langle \mathcal{I}, \sqsubseteq \rangle$ of abstract 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 for all $\mathcal{B} \in I$ there exists $\mathcal{A} \in J$ such that $\mathcal{A} \preccurlyeq \mathcal{B}$;*
- *the bottom element is the empty set $\emptyset$, the top element is the $\simeq$-equivalence class of the singleton $\{\epsilon\}$ (ϵ=empty multiset, $\epsilon \preccurlyeq \mathcal{A}$ for any $\mathcal{A} \in B_P$);*
- *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 instance, any $\mathcal{A}$ is redundant in $\{\epsilon, \mathcal{A}\}$, which, in fact, is equivalent to $\{\epsilon\}$. It is important to note that to compare two ideals we simply need to compare their generators wrt. the multiset inclusion relation $\preccurlyeq$. Thus, given a *finite* set of facts we can always remove all redundancies using a polynomial number of comparisons.

Notation.

For the sake of simplicity, in the rest of the paper we will identify an interpretation I with its class $[I]_\simeq$. Furthermore, note that if $\mathcal{A} \preccurlyeq \mathcal{B}$, then $[\![\mathcal{B}]\!] \subseteq [\![\mathcal{A}]\!]$. In contrast, if I and J are two interpretations and $I \sqsubseteq J$ then $[\![I]\!] \subseteq [\![J]\!]$.

The two relations $\preccurlyeq$ and $\sqsubseteq$ are *well-quasi orderings* [1, 14], as stated in Prop. 5.2 and Cor. 5.3 below. This property is the key point of our idea. In fact, it will allow us to prove that a *symbolic* formulation of the operator T_P working on abstract Herbrand interpretations is guaranteed to terminate on every input LO program.

PROPOSITION 5.2 (DICKSON'S LEMMA). *Let $A_1 A_2 \dots$ be an infinite sequence of multisets over the finite alphabet Σ. Then there exist two indices i and j such that $i < j$ and $A_i \preccurlyeq A_j$.*

Following [1], by definition of $\sqsubseteq$ the following Corollary holds.

COROLLARY 5.3. *There are no infinite sequences of interpretations* $I_1 I_2 \ldots I_k \ldots$ *such that for all* k *and for all* $j < k$, $I_k \not\sqsubseteq I_j$.

Corollary 5.3 ensures that it is not possible to generate infinite sequences of interpretations such that each element is not *subsumed* (using a terminology from constraint logic programming) by one of the previous elements in the sequence. The problem now is to define a fixpoint operator over abstract Herbrand interpretations that is *correct* and *complete* wrt. the ground semantics. If we find it, then we can use the corollary to prove that (for any program) its fixpoint can be reached in finitely many steps. For this purpose and using the multiset operations $\setminus$ (difference), $\bullet$ (least upper bound wrt. $\preccurlyeq$), and ϵ (empty multiset) defined in Section 2, we first define a new version of the satisfiability relation $\models$. The intuition under the judgment $I \Vdash \Delta[\mathcal{A}]$ is that $\mathcal{A}$ is the *minimal* fact (wrt. multiset inclusion) that should be added to Δ in order for $\mathcal{A} + \Delta$ to be satisfiable in I.

DEFINITION 5.4 (SATISFIABILITY). *Let* $I \in \mathcal{I}$, *then* $\Vdash$ *is defined as follows:*

$I \Vdash \top, \mathcal{A}[\epsilon]$;

$I \Vdash \mathcal{A}[\mathcal{B} \setminus \mathcal{A}]$ *for* $\mathcal{B} \in I$;

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

$I \Vdash G_1 \,\&\, G_2, \Delta[\mathcal{A}_1 \bullet \mathcal{A}_2]$ *if* $I \Vdash G_1, \Delta[\mathcal{A}_1]$, $I \Vdash G_2, \Delta[\mathcal{A}_2]$.

Given a finite interpretation I and a context Δ, the previous definition gives us an *algorithm* to compute all facts $\mathcal{A}$ such that $I \Vdash \Delta[\mathcal{A}]$ holds. Furthermore, the relation $\Vdash$ satisfies the following properties.

LEMMA 5.5. *Given* $I, J \in \mathcal{I}$,

1. *if* $I \Vdash \Delta[\mathcal{A}]$, *then* $[\![I]\!] \models \Delta[\mathcal{A}']$ *for all* $\mathcal{A}'$ *s.t.* $\mathcal{A} \preccurlyeq \mathcal{A}'$;
2. *if* $[\![I]\!] \models \Delta[\mathcal{A}']$, *then there exists* $\mathcal{A}$ *such that* $I \Vdash \Delta[\mathcal{A}]$ *and* $\mathcal{A} \preccurlyeq \mathcal{A}'$;
3. *if* $I \Vdash \Delta[\mathcal{A}]$ *and* $I \sqsubseteq J$, *then there exists* $\mathcal{A}'$ *such that* $J \Vdash \Delta[\mathcal{A}']$ *and* $\mathcal{A}' \preccurlyeq \mathcal{A}$;
4. *given a chain of abstract Herbrand interpretations* $I_1 \sqsubseteq I_2 \sqsubseteq \ldots$, *if* $[\![\bigsqcup_{i=1}^{\infty} I_i]\!] \models \Delta[\mathcal{A}]$ *then there exists* k *s.t* $[\![I_k]\!] \models \Delta[\mathcal{A}]$.

The abstract fixpoint operator $S_P : \mathcal{I} \rightsquigarrow \mathcal{I}$ should satisfy the equation $[\![S_P(I)]\!] = T_P([\![I]\!])$ (as for the S_P operator used in the symbolic semantics of CLP programs [21]). We define the new operator as follows (we recall that if $H = a_1 \parr \ldots \parr a_n$, then $\widehat{H}$ is the multiset $a_1, \ldots, a_n$).

DEFINITION 5.6 (ABSTRACT FIXPOINT OPERATOR S_P). *Given an LO program* P, *the operator* S_P *is defined as follows:*

$$S_P(I) = \{\widehat{H} + \mathcal{A} \mid H \multimapinv G \in P,\ I \Vdash G[\mathcal{A}]\}$$

The abstract (symbolic) operator S_P satisfies the following property.

PROPOSITION 5.7. *S_P is monotonic and continuous wrt.* $\sqsubseteq$.

Furthermore, the following properties show that the abstract operator is sound and complete wrt. the ground operator T_P.

PROPOSITION 5.8. *Let* $I \in \mathcal{I}$, *then* $[\![S_P(I)]\!] = T_P([\![I]\!])$.

COROLLARY 5.9. $[\![lfp(S_P)]\!] = lfp(T_P)$.

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

THEOREM 5.10 (SOUNDNESS AND COMPLETENESS). *Given an LO program* P, $O(P) = F(P) = [\![SymbF(P)]\!]$. *Furthermore, there exists* $k \in N$ *such that* $SymbF(P) = \bigsqcup_{i=0}^{k} S_P{\uparrow_k}(\emptyset)$.

Proof. Theorem 4.7 and Corollary 5.9 show that $O(P) = F(P) = [\![SymbF(P)]\!]$. Corollary 5.3 guarantees that the fixpoint of S_P can always be reached after finitely many steps. □

The previous results give us an algorithm to compute the operational and fixpoint semantics of a propositional LO program via the operator S_P. The algorithm is inspired by the *backward reachability* algorithm proposed in [1, 14] (used to compute *backwards* the closure of the *predecessor* operator of a well-structured transition system). The algorithm in pseudo-code for computing $F(P)$ is shown in Fig. 2. Cor. 5.3 guarantees that the algorithm always terminates and returns a *symbolic* representation of $O(P)$. As a corollary of Theorem 5.10, we obtain the following result.

COROLLARY 5.11. *The provability of* $P \Rightarrow G$ *in propositional LO is* decidable.

In view of Prop. 3.1, this result can be considered as an instance of the general decidability result [25] for propositional *affine* linear logic (i.e. linear logic with *weakening*).

EXAMPLE 5.12. We calculate the fixpoint semantics in a simple case. We have an LO program P made up of five clauses:

1. $a \multimapinv b \parr c$
2. $b \multimapinv (d \parr e) \,\&\, f$
3. $c \parr d \multimapinv \top$
4. $c \parr e \multimapinv c \parr b$
5. $c \parr f \multimapinv \top$

We start the computation from $S_P{\uparrow_0} = \emptyset$. The first step consists in adding the multisets corresponding to program facts, i.e., clauses 3. and 5., therefore we compute

$$S_P{\uparrow_1} = \{\{c, d\}, \{c, f\}\}$$

Procedure *symbF*(P : LO program):
set $New := \{\widehat{H} \mid H \circ\!- \top \in P\}$ **and** Old $:= \emptyset$;
repeat
set Old $:= Old \cup$ New **and** New $:= S_P$(New);
until New $\sqsubseteq$ Old;
return Old.

Figure 2: Symbolic fixpoint computation

Now, we can try to apply clauses 1., 2., and 4. to facts in $S_P{\uparrow}_1$. From the first clause, we have that $S_P{\uparrow}_1 \Vdash \{b,c\}[d]$ and $S_P{\uparrow}_1 \Vdash \{b,c\}[f]$, therefore $\{a,d\}$ and $\{a,f\}$ are elements of $S_P{\uparrow}_2$. Similarly, for clause 2. we have that $S_P{\uparrow}_1 \Vdash \{d,e\}[c]$ and $S_P{\uparrow}_1 \Vdash \{f\}[c]$, therefore we have, from the rule for $\&$, that $\{b,c\}$ belongs to $S_P{\uparrow}_2$ (we can also derive other judgments for clause 2., for instance $S_P{\uparrow}_1 \Vdash \{d,e\}[c,f]$, but it immediately turns out that all these judgments give rise to "redundant" information, i.e., facts that are subsumed by the already calculated ones). By clause 4., finally we have that $S_P{\uparrow}_1 \Vdash \{c,b\}[d]$ and $S_P{\uparrow}_1 \Vdash \{c,b\}[f]$, therefore $\{c,d,e\}$ and $\{c,e,f\}$ belong to $S_P{\uparrow}_2$, but this information is redundant. We can therefore take the following equivalence class as representative for $S_P{\uparrow}_2$:

$$S_P{\uparrow}_2 = \{\{c,d\},\{c,f\},\{a,d\},\{a,f\},\{b,c\}\}$$

We can similarly calculate $S_P{\uparrow}_3$. For clause 1. we immediately have that $S_P{\uparrow}_2 \Vdash \{b,c\}[\epsilon]$, so that $\{a\}$ is an element of $S_P{\uparrow}_3$; this makes the information given by $\{a,d\},\{a,f\}$ in $S_P{\uparrow}_2$ redundant. No additional (not redundant) elements are obtained from clause 2., while from clause 4. we can get that $\{c,e\}$ is another element of $S_P{\uparrow}_3$. We therefore have

$$S_P{\uparrow}_3 = \{\{c,d\},\{c,f\},\{b,c\},\{a\},\{c,e\}\}$$

The reader can verify that $S_P{\uparrow}_4 = S_P{\uparrow}_3 = SymbF(P)$ so that

$$O(P) = F(P) = [\![\{\{c,d\},\{c,f\},\{b,c\},\{a\},\{c,e\}\}]\!]$$

□

6. A BOTTOM-UP SEMANTICS FOR LO$_{\mathbf{1}}$

As shown in [2], the original formulation of the language LO can be extended in order to take into consideration more powerful programming constructs. In this paper we will consider an extension of LO where goal formulas range over the **G**-formulas of Section 3 and over the logical constant $\mathbf{1}$. In other words, we extend LO with clauses of the following form:

$$A_1 \parr \ldots \parr A_n \circ\!- \mathbf{1}$$

We name the resulting language LO$_{\mathbf{1}}$, and use the notation $P \Rightarrow_{\mathbf{1}} \Delta$ for LO$_{\mathbf{1}}$ sequents. The meaning of the new kind of clauses is given by the following inference scheme:

$$\dfrac{\dfrac{}{P \Rightarrow_{\mathbf{1}} \mathbf{1}}\ \mathbf{1}_r}{P \Rightarrow_{\mathbf{1}} a_1,\ldots,a_n}\ bc\ (a_1 \parr \ \ldots \parr a_n \circ\!- \mathbf{1} \ \in \ P)$$

Note that there cannot be other *resources* in the right-hand side of the lower sequent apart from $a_1,\ldots,a_n$. Thus, in contrast with $\top$, the constant $\mathbf{1}$ introduces the possibility of *counting* resources. Provability in LO$_{\mathbf{1}}$ amounts to provability in the proof system for LO augmented with the $\mathbf{1}_r$ rule.

As for LO, let us define the top-down operational semantics of an LO$_{\mathbf{1}}$ program as follows:

$$O_{\mathbf{1}}(P) = \{\mathcal{A} \mid \mathcal{A}\ is\ a\ fact\ and\ P \Rightarrow_{\mathbf{1}} \mathcal{A}\ is\ provable\}$$

Now the question is: is it still possible to find a finite representation of $O_{\mathbf{1}}(P)$? We first note that, in contrast with Prop. 3.1, the weakening rule is not admissible in LO$_{\mathbf{1}}$. This implies that we cannot use the same techniques we used for the fragment without $\mathbf{1}$. More formally, the following proposition gives a negative answer to our question.

Proposition 6.1. *Given an LO$_{\mathbf{1}}$ program P, there is no algorithm to compute $O_{\mathbf{1}}(P)$.*

Proof. To prove the result we present an encoding of Vector Addition Systems (VAS) into LO$_{\mathbf{1}}$ programs. A VAS is defined via a transition system defined over n variables $\langle x_1,\ldots,x_n\rangle$ ranging over positive integers. The transition rules have the form $x_1' = x_1+\delta_1,\ldots,x_n' = x_n+\delta_n$ where δ_n is an integer constant. Whenever $\delta_i < 0$, guards of the form $x_i \geq -\delta_i$ ensure that the variables assume only positive values. Following [10], the encoding of a VAS in LO$_{\mathbf{1}}$ is defined as follows. We associate a propositional symbol $a_i \in \Sigma$ to each variable x_i. A VAS-transition now becomes a rewriting rule $H \circ\!- B$ where $Occ_{\widehat{B}}(a_i) = -\delta_i$ if $\delta_i < 0$ (tokens removed from place i) and $Occ_{\widehat{H}}(a_i) = \delta_i$ if $\delta_i \geq 0$ (tokens added to place i). We encode the set of initial markings (i.e. assignments for the variables x_i's) $M_1,\ldots,M_k$ using k clauses as follows. The i-*th* clause $H_i \circ\!- \mathbf{1}$ is such that if M_i is the assignment $\langle x_1 = c_1,\ldots,x_n = c_n\rangle$ then $Occ_{\widehat{H_i}}(a_j) = c_j$ for $j : 1,\ldots,n$. Based on this idea, if P_V is the program that encodes the VAS V it is easy to check that $O(P_V)$ corresponds to the set of *reachable* markings of V (i.e. to the closure *post** of the *successor* operator *post* wrt. V and the initial markings). From classical results on Petri Nets [13], there is no algorithm to compute the set of reachable states of a VAS V ($=O(P_V)$). If not so, we would be able to solve the *marking equivalence* problem that is known to be undecidable [13]. □

Despite of Prop. 6.1, it is still possible to define a *symbolic*, effective fixpoint operator for LO$_{\mathbf{1}}$ programs as we will show in the following section. Before going into more details, we first rephrase the semantics of Section 4 for LO$_{\mathbf{1}}$. In the rest of the paper we will still denote the satisfiability judgments for LO$_{\mathbf{1}}$ with $\models$ and $\Vdash$.

Definition 6.2 (Satisfiability in LO$_{\mathbf{1}}$).

Let $I \subseteq B_P$, then $\models$ is defined as follows:

$I \models \mathbf{1}[\epsilon]$;

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

$I \models \mathcal{A}[\mathcal{A}']$ *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* $I \models G_2, \Delta[\mathcal{A}]$.

The new satisfiability relation satisfies the following properties.

LEMMA 6.3. *For any interpretations I, J, context Δ, and fact $\mathcal{A}$,*

i) $I \models \Delta[\mathcal{A}]$ *if and only if* $I \models \Delta, \mathcal{A}[\epsilon]$;

ii) *if* $I \subseteq J$ *and* $I \models \Delta[\mathcal{A}]$ *then* $J \models \Delta[\mathcal{A}]$;

iii) *given a chain of interpretations* $I_1 \subseteq I_2 \subseteq \ldots$, *if* $\bigcup_{i=1}^{\infty} I_i \models \Delta[\mathcal{A}]$ *then there exists k s.t.* $I_k \models \Delta[\mathcal{A}]$.

The fixpoint operator $T_P^{\mathbf{1}}$ is defined like T_P.

DEFINITION 6.4 (FIXPOINT OPERATOR $T_P^{\mathbf{1}}$). *Given an $LO_{\mathbf{1}}$ program P, the operator $T_P^{\mathbf{1}}$ is defined as follows:*

$$T_P^{\mathbf{1}}(I) = \{\widehat{H} + \mathcal{A} \mid H \multimapinv G \in P,\ I \models G[\mathcal{A}]\}$$

The following property holds.

PROPOSITION 6.5. *$T_P^{\mathbf{1}}$ is monotonic and continuous wrt. $\subseteq$.*

The fixpoint semantics is defined as $F_{\mathbf{1}}(P) = \mathit{lfp}(T_P^{\mathbf{1}}) = T_P^{\mathbf{1}}{\uparrow}_{\omega}$. It is sound and complete with respect to the operational semantics, as stated in the following theorem.

THEOREM 6.6 (SOUNDNESS AND COMPLETENESS). *For every $LO_{\mathbf{1}}$ program P, $F_{\mathbf{1}}(P) = O_{\mathbf{1}}(P)$.*

7. CONSTRAINT SEMANTICS FOR LO$_{\mathbf{1}}$

In this Section we will define a *symbolic* fixpoint operator which relies on a constraint-based representation of provable multisets. Application of this operator is effective. Prop. 6.1 shows however that there is no guarantee that its fixpoint can be reached after finitely many steps. According to the encoding of VAS used in the proof of Prop. 6.1, let $\mathbf{x} = \langle x_1, \ldots, x_n\rangle$ be a vector of variables, where variable x_i denotes the number of occurrences of $a_i \in \Sigma$ in a given fact. Then we can immediately recover the semantics of Section 5 using a very simple class of linear constraints. Namely, given a fact $\mathcal{A}$ we can denote its closure, i.e., the ideal $[\![\mathcal{A}]\!]$, by the constraint

$$\varphi_{[\![\mathcal{A}]\!]} \ \equiv\ \bigwedge_{i=1}^{n} x_i \geq Occ_{\mathcal{A}}(a_i).$$

Then all the operations on multisets involved in the definition of S_P (see Def. 5.4) can be expressed as operations over linear constraints. In particular, given the ideals $[\![\mathcal{A}]\!]$ and $[\![\mathcal{B}]\!]$, the ideal $[\![\mathcal{A} \bullet \mathcal{B}]\!]$ is represented as the constraint

$$\varphi_{[\![\mathcal{A}\bullet\mathcal{B}]\!]} = \varphi_{[\![\mathcal{A}]\!]} \wedge \varphi_{[\![\mathcal{B}]\!]},$$

while $[\![\mathcal{B} \setminus \mathcal{A}]\!]$, for a given multiset $\mathcal{A}$, is represented as the constraint

$$\varphi_{[\![\mathcal{B}\setminus\mathcal{A}]\!]} = \exists \mathbf{x}'.(\varphi_{[\![\mathcal{B}]\!]}[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathcal{A}}(\mathbf{x}, \mathbf{x}')),$$

where

$$\rho_{\mathcal{A}}(\mathbf{x}, \mathbf{x}') \ \equiv\ \bigwedge_{i=1}^{n} x_i = x_i' - Occ_{\mathcal{A}}(a_i) \ \wedge\ x_i \geq 0.$$

The constraint $\rho_{\mathcal{A}}$ models the removal of the occurrences of $\mathcal{A}$ from all elements of the denotation of $\mathcal{B}$. Similarly, $[\![\mathcal{B} + \mathcal{A}]\!]$, for a given multiset $\mathcal{A}$, is represented as the constraint

$$\varphi_{[\![\mathcal{B}+\mathcal{A}]\!]} = \exists \mathbf{x}'.(\varphi_{[\![\mathcal{B}]\!]}[\mathbf{x}'/\mathbf{x}] \wedge \alpha_{\mathcal{A}}(\mathbf{x}, \mathbf{x}')),$$

where

$$\alpha_{\mathcal{A}}(\mathbf{x}, \mathbf{x}') \ \equiv\ \bigwedge_{i=1}^{n} x_i = x_i' + Occ_{\mathcal{A}}(a_i).$$

In order to give a semantics for LO$_{\mathbf{1}}$, we need to add a class of constraints for representing multisets which are not upward-closed (i.e. which are not ideals). This is due to the fact that introducing the constant $\mathbf{1}$ breaks down the monotonicity property given by Prop. 3.1, so that the abstraction based on ideals cannot be used anymore. Given a multiset $\mathcal{A}$, we can simply represent it as the linear constraint

$$\varphi_{\mathcal{A}} \ \equiv\ \bigwedge_{i=1}^{n} x_i = Occ_{\mathcal{A}}(a_i).$$

The operations over linear constraints discussed previously extend smoothly when adding this new class of equality constraints. In particular, given two constraints φ and ψ, their conjunction $\varphi \wedge \psi$ still plays the role that the operation $\bullet$ (least upper bound of multisets) had in Def. 5.4, while $\exists \mathbf{x}'.(\varphi[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathcal{A}}(\mathbf{x}, \mathbf{x}'))$, for a given multiset $\mathcal{A}$, plays the role of multiset difference. The reader can compare Def. 5.4 with Def. 7.2. Based on these ideas, we can define a bottom-up evaluation procedure for LO$_{\mathbf{1}}$ programs via an extension $S_P^{\mathbf{1}}$ of the operator S_P.

In the following we will use the notation $\widehat{\mathbf{c}}$, where $\mathbf{c} = \langle c_1, \ldots, c_n\rangle$ is a solution of a constraint φ (i.e. an assignment of natural numbers to the variables $\mathbf{x}$ which satisfies φ), to indicate the multiset over $\Sigma = \{a_1, \ldots, a_n\}$ which contains c_i occurrences of every propositional symbol a_i (i.e. according to the notation introduced above, $\mathbf{c}$ is the unique solution of $\varphi_{\widehat{\mathbf{c}}}$). We extend this definition to a set C of constraint solutions by $\widehat{C} = \{\widehat{\mathbf{c}} \mid \mathbf{c} \in C\}$. We then define the denotation of a given constraint φ, written $[\![\varphi]\!]_{\mathbf{1}}$, as the set of multisets corresponding to solutions of φ, i.e., $[\![\varphi]\!]_{\mathbf{1}} = \{\widehat{\mathbf{c}} \mid \mathbf{x} = \mathbf{c}$ satisfies $\varphi\}$.

We introduce an equivalence relation $\simeq$ over constraints, given by $\varphi \simeq \psi$ if and only if $[\![\varphi]\!]_{\mathbf{1}} = [\![\psi]\!]_{\mathbf{1}}$, i.e., we identify constraints with the same set of solutions. For the sake of simplicity, in the following we will identify a constraint with its equivalence class, i.e., we will simply write φ instead of $[\varphi]_{\simeq}$.

Let LC_Σ be the set of (equivalence classes of) of linear constraints over the variables $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ associated to the signature $\Sigma = \{a_1, \ldots, a_n\}$. The operator $S^{\mathbf{1}}_P$ is defined on *constraint interpretations* consisting of sets (disjunctions) of (equivalence classes of) linear constraints. The denotation $[\![I]\!]_{\mathbf{1}}$ of a constraint interpretation I extends the one for constraints as expected: $[\![I]\!]_{\mathbf{1}} = \{[\![\varphi]\!]_{\mathbf{1}} \mid \varphi \in I\}$. Interpretations form a complete lattice with respect to set inclusion.

DEFINITION 7.1 (CONSTRAINT INTERPRETATION). *We say that $I \subseteq LC_\Sigma$ is a constraint interpretation. Constraint interpretations form a complete lattice $\mathcal{C} = \langle \mathcal{P}(LC_\Sigma), \subseteq \rangle$ with respect to set inclusion.*

We obtain then a new notion of satisfiability using operations over constraints as follows. In the following definitions we assume that the conditions apply only when the constraints are **satisfiable** (e.g. $x = 0 \wedge x \geq 1$ has no solutions thus the following rules cannot be applied to this case).

DEFINITION 7.2 (SATISFIABILITY IN $\mathrm{LO}_{\mathbf{1}}$). *Let $I \in \mathcal{C}$, then $\Vdash$ is defined as follows:*

$I \Vdash \mathbf{1}[\varphi]$ *where* $\varphi \;\equiv\; x_1 = 0 \wedge \ldots \wedge x_n = 0$;

$I \Vdash \top, \mathcal{A}[\varphi]$ *where* $\varphi \;\equiv\; x_1 \geq 0 \wedge \ldots \wedge x_n \geq 0$;

$I \Vdash \mathcal{A}[\varphi]$ *where* $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathcal{A}}(\mathbf{x}, \mathbf{x}')),\ \psi \in I$;

$I \Vdash G_1 \mathbin{⅋} G_2, \Delta[\varphi]$ *if* $I \Vdash G_1, G_2, \Delta[\varphi]$;

$I \Vdash G_1 \,\&\, G_2, \Delta[\varphi_1 \wedge \varphi_2]$ *if* $I \Vdash G_1, \Delta[\varphi_1],\ I \Vdash G_2, \Delta[\varphi_2]$.

The relation $\Vdash$ satisfies the following properties.

LEMMA 7.3. *Given $I, J \in \mathcal{C}$,*

1. *if $I \Vdash \Delta[\varphi]$, then $[\![I]\!]_{\mathbf{1}} \models \Delta[\mathcal{A}]$ for every $\mathcal{A} \in [\![\varphi]\!]_{\mathbf{1}}$;*
2. *if $[\![I]\!]_{\mathbf{1}} \models \Delta[\mathcal{A}]$, then there exists φ such that $I \Vdash \Delta[\varphi]$ and $\mathcal{A} \in [\![\varphi]\!]_{\mathbf{1}}$;*
3. *if $I \subseteq J$ and $I \Vdash \Delta[\varphi]$, then $J \Vdash \Delta[\varphi]$;*
4. *given a chain of constraint interpretations $I_1 \subseteq I_2 \subseteq \ldots$, if $\bigcup_{i=1}^{\infty} I_i \Vdash \Delta[\varphi]$ then there exists k s.t. $I_k \Vdash \Delta[\varphi]$.*

We are now ready to define the extended operator $S^{\mathbf{1}}_P$.

DEFINITION 7.4 (SYMBOLIC FIXPOINT OPERATOR $S^{\mathbf{1}}_P$). *Given an $LO_{\mathbf{1}}$ program P, the operator $S^{\mathbf{1}}_P$ is defined as follows:*

$$S^{\mathbf{1}}_P(I) = \{\ \varphi \quad \mid \quad H \mathrel{\circ\!-} G \in P,\ I \Vdash G[\psi],\ \varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \alpha_{\widehat{H}}(\mathbf{x}, \mathbf{x}'))\}$$

The new operator satisfies the following property.

PROPOSITION 7.5. *The operator $S^{\mathbf{1}}_P$ is monotonic and continuous over the lattice $\mathcal{C}$.*

Furthermore, it is a symbolic version of the ground operator $T^{\mathbf{1}}_P$, as stated below.

PROPOSITION 7.6. *Let $I \in \mathcal{C}$, then $[\![S^{\mathbf{1}}_P(I)]\!]_{\mathbf{1}} = T^{\mathbf{1}}_P([\![I]\!]_{\mathbf{1}})$.*

COROLLARY 7.7. *$[\![\mathit{lfp}(S^{\mathbf{1}}_P)]\!]_{\mathbf{1}} = \mathit{lfp}(T^{\mathbf{1}}_P)$.*

Now, let $\mathit{SymbF}_{\mathbf{1}}(P) = \mathit{lfp}(S^{\mathbf{1}}_P)$, then we have the following main theorem that shows that $S^{\mathbf{1}}_P$ can be used (without termination guarantee) to compute symbolically the set of logical consequences of an $\mathrm{LO}_{\mathbf{1}}$ program.

THEOREM 7.8 (SOUNDNESS AND COMPLETENESS). *Given an $LO_{\mathbf{1}}$ program P, $O_{\mathbf{1}}(P) = F_{\mathbf{1}}(P) = [\![\mathit{SymbF}_{\mathbf{1}}(P)]\!]_{\mathbf{1}}$.*

8. BOTTOM-UP EVALUATION FOR $\mathrm{LO}_{\mathbf{1}}$

Using a constraint-based representation for $\mathrm{LO}_{\mathbf{1}}$ provable multisets, we have reduced the problem of computing $O_{\mathbf{1}}(P)$ to the problem of computing the reachable states of a system with *integer* variables. As shown by Prop. 6.1, the termination of the algorithm is not guaranteed a priori. In this respect, Theorem 5.10 gives us sufficient conditions that ensure its termination.

The symbolic fixpoint operator $S^{\mathbf{1}}_P$ introduced in section 7 is defined over the lattice $\mathcal{C} = \langle \mathcal{P}(LC_\Sigma), \subseteq \rangle$, with set inclusion being the partial order relation and set union the least upper bound operator. When we come to a concrete implementation of $S^{\mathbf{1}}_P$, it is worth considering a weaker ordering relation between interpretations, namely pointwise *subsumption*. Let $\preccurlyeq$ be the partial order between (equivalence classes of) constraints given by $\varphi \preccurlyeq \psi$ if and only if $[\![\psi]\!]_{\mathbf{1}} \subseteq [\![\varphi]\!]_{\mathbf{1}}$. Then we say that an interpretation I is subsumed by an interpretation J, written $I \sqsubseteq J$, if and only if for every $\varphi \in I$ there exists $\psi \in J$ such that $\psi \preccurlyeq \varphi$.

As we do not need to distinguish between different interpretations representing the same set of solutions, we can consider interpretations I and J to be equivalent in case both $I \sqsubseteq J$ and $J \sqsubseteq I$ hold. In this way, we get a lattice of interpretations ordered by $\sqsubseteq$ and such that the least upper bound operator is still set union. This construction is the natural extension of the one of Section 5. Actually, when we limit ourselves to considering LO programs (i.e. without the constant $\mathbf{1}$) it turns out that we need only consider constraints of the form $\mathbf{x} \geq \mathbf{c}$, which can be abstracted away by considering the upward closure of $\widehat{\mathbf{c}}$, as we did in Section 5. The reader can note that the $\preccurlyeq$ relation defined above for constraints is an extension of the multiset inclusion relation we used in Section 5.

The construction based on $\sqsubseteq$ can be directly incorporated into the semantic framework presented in Section 7, where, for the sake of simplicity, we have adopted an approach based on $\subseteq$. Of course, relation $\subseteq$ is stronger than $\sqsubseteq$, therefore a computation based on $\sqsubseteq$ is correct and it terminates every time a computation based on $\subseteq$ does. However, the converse does not always hold, and this is why a concrete algorithm for computing the least fixpoint of $S_P^{\mathbf{1}}$ relies on subsumption. Let us see an example.

EXAMPLE 8.1. We calculate the fixpoint semantics for the following $LO_{\mathbf{1}}$ program made up of six clauses:

1. $a \circ\!- \mathbf{1}$
2. $a \parr b \circ\!- \top$
3. $c \parr c \circ\!- \top$
4. $b \parr b \circ\!- a$
5. $a \circ\!- b$
6. $c \circ\!- a \,\&\, b$

Let $\Sigma = \{a, b, c\}$ and consider constraints over the variables $\mathbf{x} = \langle x_a, x_b, x_c\rangle$. We have that $S_P{\uparrow}_0 = \emptyset \Vdash \mathbf{1}[x_a = 0 \wedge x_b = 0 \wedge x_c = 0]$, therefore, by the first clause, $\varphi \in S_P{\uparrow}_1$, where $\varphi = \exists \mathbf{x}'.(x'_a = 0 \wedge x'_b = 0 \wedge x'_c = 0 \wedge x_a = x'_a + 1 \wedge x_b = x'_b \wedge x_c = x'_c)$, which is equivalent to $x_a = 1 \wedge x_b = 0 \wedge x_c = 0$. From now on, we leave to the reader the details concerning equivalence of constraints. By reasoning in a similar way, using clauses 2. and 3. we calculate $S_P{\uparrow}_1$ (see Fig. 3).

We now compute $S_P{\uparrow}_2$. By 4., as $S_P{\uparrow}_1 \Vdash a[x_a = 0 \wedge x_b = 0 \wedge x_c = 0]$, we get $x_a = 0 \wedge x_b = 2 \wedge x_c = 0$, and, similarly, we get $x_a \geq 0 \wedge x_b \geq 3 \wedge x_c \geq 0$. By 5., we have $x_a \geq 2 \wedge x_b \geq 0 \wedge x_c \geq 0$, while clause 6. is not (yet) applicable. Therefore, modulo redundant constraints (i.e. constraints *subsumed* by the already calculated ones) the value of $S_P{\uparrow}_2$ is given in Fig. 3.

Now, we can compute $S_P{\uparrow}_3$. By 4. and $x_a \geq 2 \wedge x_b \geq 0 \wedge x_c \geq 0 \in S_P{\uparrow}_2$ we get $x_a \geq 1 \wedge x_b \geq 2 \wedge x_c \geq 0$, which is subsumed by $x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0$. By 5. and $x_a = 0 \wedge x_b = 2 \wedge x_c = 0$, we get $x_a = 1 \wedge x_b = 1 \wedge x_c = 0$, subsumed by $x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0$. Similarly, by 5. and $x_a \geq 0 \wedge x_b \geq 3 \wedge x_c \geq 0$ we get redundant information. By 6., from $x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0$ and $x_a = 0 \wedge x_b = 2 \wedge x_c = 0$ we get $x_a = 0 \wedge x_b = 1 \wedge x_c = 1$, from $x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0$ and $x_a \geq 0 \wedge x_b \geq 3 \wedge x_c \geq 0$ we get $x_a \geq 0 \wedge x_b \geq 2 \wedge x_c \geq 1$, and finally from $x_a \geq 2 \wedge x_b \geq 0 \wedge x_c \geq 0$ and $x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0$ we have $x_a \geq 1 \wedge x_b \geq 0 \wedge x_c \geq 1$. The reader can verify that no additional provable multisets can be obtained. It is somewhat tedious, but in no way difficult, to verify that clause 6. yields only redundant information when applied to every possible couple of constraints in $S_P{\uparrow}_3$. We have then $S_P{\uparrow}_4 = S_P{\uparrow}_3 = \mathit{SymbF}_{\mathbf{1}}(P)$, so that in this particular case we achieve termination. We can reformulate the operational semantics of P using the more suggestive multiset notation (we recall that $[\![\mathcal{A}]\!] = \{\mathcal{B} \mid \mathcal{A} \preccurlyeq \mathcal{B}\}$, where $\preccurlyeq$ is multiset inclusion):

$$O_{\mathbf{1}}(P) = F_{\mathbf{1}}(P) = \{\{a\}, \{b, b\}, \{b, c\}\} \cup [\![\{a, b\}, \{c, c\}, \{b, b, b\}, \{a, a\}, \{b, b, c\}, \{a, c\}]\!]$$

$\square$

It is often not the case that the symbolic computation of $LO_{\mathbf{1}}$ program semantics can be carried out in a finite number of steps. Nevertheless, it is important to remark that viewing the bottom-up evaluation of $LO_{\mathbf{1}}$ programs as a least fixpoint computation over infinite-state *integer* systems allows us to apply techniques and tools developed in *infinite-state* model checking (see e.g. [1, 9, 12, 14, 19]) and program analysis [11] to compute approximations of the least fixpoint of $S_P^{\mathbf{1}}$.

9. RELATED WORKS

Our work is inspired to the general decidability results for infinite-state systems based on the theory of well-quasi orderings given in [1, 14]. In fact, the construction of the least fixpoint of S_P and $S_P^{\mathbf{1}}$ can be viewed as an instance of the *backward reachability* algorithm for transition systems presented in [1]. Differently from [1, 14], we need to add special rules (via the satisfiability relation $\Vdash$) to handle formulas with the connectives $\&$, $\top$ and $\mathbf{1}$.

Other sources of inspiration come from linear logic programming. In [18], Harland and Winikoff present an abstract deductive system for the 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. The satisfiability relations we use in the definition of the fixpoint operators correspond to top-down steps within their bottom-up evaluation scheme. The framework is given for a more general fragment than LO. However, they do not provide an *effective* fixpoint operator as we did in the case of LO and $LO_{\mathbf{1}}$, and they do not discuss computability issues for the resulting derivability relation.

In [6], Andreoli, Pareschi and Castagnetti present a partial evaluation scheme for propositional LO (i.e without $\mathbf{1}$). Given an initial goal G, they use a construction similar to Karp and Miller's *coverability graph* [22] for Petri Nets to build a finite representation of a proof tree for G. During the *top-down* construction of the graph for G, they apply in fact a *generalization step* that works as follows. If a goal, say $\mathcal{B}$, that has to be proved is subsumed by a node already visited, say $\mathcal{A}$, (i.e. $\mathcal{B} = \mathcal{A} + \mathcal{A}'$), then the part of proof tree between the two goals is replaced by a proof tree for $\mathcal{A} + (\mathcal{A}')^*$; $\mathcal{A} + (\mathcal{A}')^*$ is a finite representation of the union of $\mathcal{A}$ with the closure of $\mathcal{A}'$. They use Dickson's Lemma to show that the construction always terminates. In the case of LO, the main difference with our approach is that we give a goal independent *bottom-up* algorithm. Technically, another difference is that in our fixpoint semantics we do not need any *generalization* step. In fact, in our setting the computation starts directly from (a representation of) *upward-closed* sets of contexts. This simplifies the computation as shown in Example 5.12 (we only need to test $\preccurlyeq$). Finally, differently from [6], in this paper we have given also a formal semantics for the extension of LO with the constant $\mathbf{1}$.

The partial evaluation scheme of [6] is aimed at compile-time optimizations of abstractions of *Linlog* programs. Another example of analysis of concurrent languages based on linear logic is given in [23], where the authors present a type inference procedure that returns an approximation of the number of messages exchanged by HACL processes.

In [10] Cervesato shows how to encode Petri Nets in LO, Lolli and Forum by exploiting the different features of these languages. We use some of these ideas to prove Prop. 6.1.

Finally, our semantics for LO shares some similarities with the bottom-up semantics for *disjunctive logic programs* of

$$
\begin{array}{rcll}
S_P\uparrow_1 & = & \{ & x_a = 1 \wedge x_b = 0 \wedge x_c = 0, \quad x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0, \quad x_a \geq 0 \wedge x_b \geq 0 \wedge x_c \geq 2 \quad \} \\
S_P\uparrow_2 & = & \{ & x_a = 1 \wedge x_b = 0 \wedge x_c = 0, \quad x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0, \quad x_a \geq 0 \wedge x_b \geq 0 \wedge x_c \geq 2, \\
& & & x_a = 0 \wedge x_b = 2 \wedge x_c = 0, \quad x_a \geq 0 \wedge x_b \geq 3 \wedge x_c \geq 0, \quad x_a \geq 2 \wedge x_b \geq 0 \wedge x_c \geq 0 \quad \} \\
S_P\uparrow_3 & = & \{ & x_a = 1 \wedge x_b = 0 \wedge x_c = 0, \quad x_a \geq 1 \wedge x_b \geq 1 \wedge x_c \geq 0, \quad x_a \geq 0 \wedge x_b \geq 0 \wedge x_c \geq 2, \\
& & & x_a = 0 \wedge x_b = 2 \wedge x_c = 0, \quad x_a \geq 0 \wedge x_b \geq 3 \wedge x_c \geq 0, \quad x_a \geq 2 \wedge x_b \geq 0 \wedge x_c \geq 0, \\
& & & x_a = 0 \wedge x_b = 1 \wedge x_c = 1, \quad x_a \geq 0 \wedge x_b \geq 2 \wedge x_c \geq 1, \quad x_a \geq 1 \wedge x_b \geq 0 \wedge x_c \geq 1 \quad \}
\end{array}
$$

Figure 3: Symbolic fixpoint computation for the program in Example 8.1

Minker, Rajasekar and Lobo [29]. In a *disjunctive logic program*, the head of a clause is a disjunction of atomic formulas, whereas the body is a conjunction of atomic formulas. In the semantics of [29] interpretations are collections of *sets* (disjunctions) of atomic formulas. Only minimal (wrt. set inclusion) sets are kept at each fixpoint iteration. In contrast, in our setting we need to consider collections of *multisets* of formulas. Therefore, in the propositional case in order to prove the convergence of the fixpoint iteration, we need an argument (Dickson's lemma) stronger than the finiteness of the *extended* Herbrand base of [29] (collection of all minimal sets).

10. CONCLUSIONS AND FUTURE WORK

In this preliminary work we have defined a bottom-up semantics for LO [4] enriched with the constant **1**. In the propositional case, we have shown that without **1** the fixpoint semantics is finitely computable. Our fixpoint operator is defined over constraints and gives us an effective way to evaluate bottom-up (abstractions of) linear logic programs.

To conclude, let us discuss the directions of research related to our work that we find more promising.

Linear Logic Programming. It would be interesting to extend the techniques we presented in this paper to larger fragments of linear logic. In particular, it would be interesting to define a bottom-up evaluation for languages like Lolli[20] and Lygon[17], and to study techniques for first-order formulation for all these languages.

Verification. In [12], Delzanno and Podelski show that properties of concurrent systems expressed in *temporal logic* can be defined in terms of fixpoint semantics of logic programs. To give some ideas about this connection, let us note that we can check if a *safety property* $\Box F$ holds (i.e. F holds in every reachable state), exploring the set of states of the transition system taken into consideration that are backward reachable from the set of states that don't satisfy F. This procedure corresponds to the computation of the fixpoint semantics of a logic program that encodes a concurrent system (e.g. Petri Nets in Theorem 6.1) starting from a set of facts that encodes F. Note that in [12] synchronization between processes is achieved via *shared variables*, whereas in linear logic synchronization can be expressed via multiple headed clauses. Thus, our semantics might be a first step towards the extension of the metaphor of [12] to concurrent systems in which synchronization is expressed at the logical level. The other way round, through the connection between semantics and verification, techniques used for infinite-state systems with integer variables (see e.g. [12, 9, 19]) can be re-used in order to compute a static analysis of linear logic programs.

Proof Theory. The connection we establish in this paper indicates a potential connection between the general decidability results for infinite-state systems of [1, 14] and provability in *sub-structural* logics like LO and *affine* linear logic. Viewing the provability relation as a transition relation, it might be possible to find a notion of *well-structured* proof system (paraphrasing the notion of *well-structured* transition systems of [1, 14]), i.e., a general notion of provability that ensures the termination of the bottom-up generation of valid formulas.

11. ACKNOWLEDGMENTS

The authors would like to thank Maurizio Gabbrielli for his encouragement, and the anonymous referees for helpful comments and for pointing out to us the reference [29] we were not aware of.

12. REFERENCES

[1] 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.

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

[3] J. M. Andreoli. Coordination in LO. In *Coordination Programming: Mechanisms, Models and Semantics.* Imperial College Press, London, 1996.

[4] J. M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-In Inheritance. *New Generation Computing*, 9(3+4):445-473, 1991.

[5] J. M. Andreoli and R. Pareschi. Communication as Fair Distribution of Knowledge. In *Proc. of OOPSLA '91*, pages 212-229, 1991.

[6] J. M. Andreoli, R. Pareschi and T. Castagnetti. Static Analysis of Linear Logic Programming. *New Generation Computing*, 15(4), 1997.

[7] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-Semantics Approach: Theory and Applications. *Journal of Logic Programming* 19-20: 149-197, 1994.

[8] M. Bozzano, G. Delzanno and M. Martelli. A Bottom-up Semantics for LO: Preliminary Results. Technical Report DISI-00-06, Università di Genova, March 2000.

[9] T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite-state Systems using Presburger Arithmetics. In *Proc. of CAV'97*, LNCS 1254, pages 400-411. Springer-Verlag, 1997.

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

[11] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *Proc. of POPL'78*, pages 84-96, 1978.

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

[13] J. Esparza and M. Nielsen. Decibility Issues for Petri Nets - a Survey. *Journal of Informatic Processing and Cybernetics*, 30(3):143-160, 1994.

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

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

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

[17] J. A. Harland, D. Pym, and M. Winikoff. Programming in Lygon: An Overview. In *Proc. of AMAST'96*, LNCS 1101, pages 391-405, 1996.

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

[19] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a Model Checker for Hybrid Systems. In *Proc. of CAV'97*, LNCS 1254, pages 460-463, 1997.

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

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

[22] R. M. Karp and R. E. Miller. Parallel Program Schemata. *Journal of Computer and System Sciences*, 3, pages 147-195, 1969.

[23] N. Kobayashi, M. Nakade, and A. Yonezawa. Static Analysis of Communication for Asynchronous Concurrent Programming Languages. In *Proc. of SAS'95*, LNCS 983, pages 225-242, 1995.

[24] N. Kobayashi and A. Yonezawa. Asynchronous Communication Model based on Linear Logic. *Formal Aspects of Computing*, 7:113-149, 1995.

[25] A. P. Kopylov. Propositional Linear Logic with Weakening is Decidable. In *Proc. of the 10th Annual IEEE Symposium on Logic in Computer Science*, San Diego, California, 1995.

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

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

[28] D. Miller. A Survey of Linear Logic Programming. *Computational Logic: The Newsletter of the European Network in Computational Logic*, 2(2):63-67, 1995.

[29] J. Minker, A. Rajasekar, and J. Lobo. Theory of Disjunctive Logic Programs. In *Computational Logic: Essays in honor of Alan Robinson*. MIT Press, 1991.