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.

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 [5].

Technically, our contributions are as follows. We first consider a formulation of LO with $\otimes$, $\alpha$, $\&$ 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 $\Delta$ is provable in $P$, then any $\Delta'$ such that $\Delta$ is a sub-multiset of $\Delta'$ 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 $S^p_T$ 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 $S^p_T$ 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 1 to the original formulation of LO in [4] breaks down the above mentioned property. Nevertheless, it is still possible to define an effective $S^p_T$ 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 $S^p_T$. 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 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 $S^p_T$ and prove that the least fixpoint of $S^p_T$ characterizes the operational semantics of a program. In Section 5 we introduce the symbolic $S^p_T$ operator and we relate it to $S^p_T$. In Section 6 we consider an extended fragment with 1, extending the notion of satisfiability given in Section 4 and introducing an operator $T^p_D$. In Section 7 we introduce a symbolic operator $S_D^p$ 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 $A, B, C, \ldots$ to denote multisets of propositional symbols, hereafter called facts, defined over a fixed signature $\Sigma = \{a_1, \ldots, a_n\}$. A fact $A$ is uniquely determined by a finite map $Occ : \Sigma \rightarrow \mathbb{N}$ such that $Occ(a_i)$ is the number of occurrences of $a_i$ in $A$. Facts are ordered according to the multiset inclusion relation $\preceq$ as defined follows. $A \preceq B$ if and only if $Occ_A(a_i) \leq Occ_B(a_i)$ for $i = 1, \ldots, n$. The empty multiset is denoted $\emptyset$ and is such that $Occ_A(a_i) = 0$ for $i = 1, \ldots, n$, and $\emptyset \preceq A$ for any $A$. The multiset union $A \cup B$ (alternatively $A + B$ when '+' is ambiguous) of two facts $A$ and $B$ is such that $Occ(A \cup B)(a_i)$ is the number of occurrences of $a_i$ in $A$ or $B$. Finally, we define a special operator $\bullet$ to compute the least upper bound of two facts with respect to $\preceq$. Namely, $A \bullet B$ is such that $Occ_A \bullet_B (a_i) = \max(0, Occ_A(a_i) - Occ_B(a_i))$ for $i = 1, \ldots, n$. In the rest of the paper we will use $\emptyset, \Theta, \ldots$ to denote multisets of possibly compound formulas. Given two multisets $\Delta$ and $\Theta$, $\Delta \preceq \Theta$ indicates multiset inclusion and $\Delta, \Theta$ multiset union, as before, and, $(\Delta, \emptyset)$ is written simply $\Delta, \emptyset$. 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 $H = a_1 \not\preceq \ldots \not\preceq a_n$, we introduce the notation $\hat{H}$ to denote the multiset $a_1, \ldots, a_n$. Finally, let $\{T \vdash \}$ be an operator defined over a complete lattice $(\mathcal{I}, \preceq)$. We define $T_{\emptyset} = \emptyset$, where $\emptyset$ is the bottom element, $T_{\{a\}} = \{T_{\emptyset}\}$ for all $a$, and $T_{\{a\} \cup \{b\}} = \{T_{\emptyset}\}$, where $\{\}$ is the least upper bound wrt. $\preceq$. Furthermore, we use $\lfp(T)$ 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 $\&$, $\otimes$, $\exists$, and $\top$. In the propositional case LO consists of the following class of formulas:

$$D ::= A_1 \otimes \ldots \otimes A_n \quad \emptyset \quad G | D \& D$$

$$G ::= G \otimes G | G & G | A | \top$$

Here $A_1, \ldots, A_n$ and $A$ range over propositional symbols from a fixed signature $\Sigma$. G-formulas correspond to goals to be evaluated in a given program. D-formulas correspond to multiple-headed program clauses. An LO program is a D-formula. Let $P$ be the program $C_1 \& \ldots \& C_n$. The execution of a multiset of 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

$$!C_1 \otimes \ldots \otimes !C_n \Rightarrow G_1 \otimes \ldots \otimes G_k.$$ 

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, $A$ denotes a multiset of atomic formulas, whereas $\Delta$ denotes a multiset of G-formulas. A sequent is provable if all branches of its proof tree terminate with instances of the $\top$ 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 $bc$
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

\[ a_1 \not\in b \ldots \not\in a_n \vdash \top \]

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 \( A \), as shown in the following scheme:

\[
\frac{P \Rightarrow \top, A}{P \Rightarrow a_1, \ldots, a_n, A} \quad \text{bc} \quad (a_1 \not\in b \ldots \not\in a_n \vdash \top \in P)
\]

This observation leads us to the following property.

**Proposition 3.1.** Given an LO program \( P \) and two contexts \( \Delta, \Delta' \) such that \( \Delta \models \Delta' \), if \( P \models \Delta \) then \( P \models \Delta' \).

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. \( a \vdash (b \land c) \not\in e \)
2. \( b \vdash \top \)
3. \( c \vdash d \not\in b \)

and consider an initial goal \( a \). Using clause 1., to prove \( a \) we have to prove \((b \land c) \not\in e\), and then, by LO \( \not\in e \) 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 \( a \).

\[ \square \]

### 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 \( \not\in \) 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 \( p \vdash G \) to the original program and analyze \( p \). For this purpose, we define the operational semantics of an LO program \( P \) as follows:

\[ O(P) = \{ A | A \text{ is a fact and } P \models A \text{ is provable} \} \]

In the rest of the paper we will always consider propositional LO programs defined over a finite set of propositional symbols \( \Sigma \). We give the following definitions.

**Definition 4.1 (Herbrand Base \( B_P \)).** Given a propositional LO program \( P \) defined over \( \Sigma \),

\[ B_P = \{ A | A \text{ is a multiset (fact) over } \Sigma \} \]

**Definition 4.2 (Herbrand Interpretation).** We say that \( I \subseteq B_P \) is a Herbrand interpretation. Herbrand interpretations form a complete lattice \( D = (\mathcal{P}(B_P), \subseteq) \) 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 \( \Delta \) in a given interpretation \( I \). For this purpose, we introduce the judgment \( I \models \Delta[A] \). In \( I \models \Delta[A] \), the output \( A \) is a fact such that \( A + \Delta \) is valid in \( I \). This notion of satisfiability is modeled according to the right-introduction rules of the connectives. The notion of output fact \( A \) 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 \( I \models A \) is defined as follows:

- \( I \models \top, A[A'] \) for any fact \( A' \);  
- \( I \models A[A'] \) if \( A + A' \in I \);  
- \( I \models G_1 \not\in G_2, \Delta[A] \) if \( I \models G_1, G_2, \Delta[A] \);  
- \( I \models G_1 & G_2, \Delta[A] \) if \( I \models G_1, \Delta[A] \) and \( I \models G_2, \Delta[A] \).

The relation \( I \models \) satisfies the following properties.

**Lemma 4.4.** For any interpretations \( I, J, \text{ context } \Delta, \text{ and fact } A \),

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

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:

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

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 $lfp(T_P) = T_P[\omega]$.

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 $P_1$ as the program $\{ A \vdash \top \mid A \in I \}$, the definition of $T_P$ can be viewed as the following instance of the cut rule of linear logic:

$$\frac{\mid P, G \vdash H, \mid P_1 \vdash G, A}{\mid P, P_1 \vdash H, A} \text{ cut}$$

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

$$\frac{P \Rightarrow H \vdash G \quad P_1 \Rightarrow G, A}{P \cup P_1 \Rightarrow H, A} \text{ cut}$$

Note that, since $H \vdash G \in P$, the sequent $P \Rightarrow H \vdash G$ 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 $H \vdash G \in P$ and for every fact $A$, $I \models G[A]$ implies $I \models H[A]$.

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 $a \vdash \top$. Then, $T_P(\emptyset)$ contains all multisets with at least one occurrence of $a$. In other words, $T_P(\emptyset) = \{ B \mid a \in B \}$, where $\subseteq$ 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 $A$ may be used to implicitly represent the ideal generated by $A$, i.e., the subset of $B_P$ defined as follows:

$$[A] = \{ B \mid A \subseteq B \}$$

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

**Definition 5.1 (Abstract Interpretation).** The lattice $(\mathcal{I}, \subseteq)$ of abstract Herbrand interpretations is defined as follows:

- $\mathcal{I} = \mathcal{P}(B_P)/\simeq$ where $I \simeq J$ if and only if $[I] = [J]$;
- $[I] \subseteq [J]$ if and only if for all $B \in I$ there exists $A \in J$ such that $A \leq B$;
- the bottom element is the empty set $\emptyset$, the top element is the $\simeq$-equivalence class of the singleton $\{\epsilon\}$ ($\epsilon$ = empty multiset, $\epsilon \leq A$ for any $A \in B_P$);
- the least upper bound $I \cup J$ is the $\simeq$-equivalence class of $I \cup J$.

The equivalence $\simeq$ allows us to reason modulo redundancies. For instance, any $A$ is redundant in $\{\epsilon, A\}$, 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 $\leq$. 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 $A \leq B$, then $[B] \subseteq [A]$. In contrast, if $I$ and $J$ are two interpretations and $I \subseteq J$ then $[I] \subseteq [J]$.

The two relations $\leq$ and $\subseteq$ 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 $A_1, A_2, \ldots$ be an infinite sequence of multisets over the finite alphabet $\Sigma$. Then there exist two indices $i$ and $j$ such that $i < j$ and $A_i \leq A_j$. 

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. \( \preceq \)), and \( \epsilon \) (empty multiset) defined in Section 2, we first define a new version of the satisfiability relation \( \models \). The intuition under the judgment \( I \models \Delta[A] \) is that \( A \) is the minimal fact (wrt. multiset inclusion) that should be added to \( \Delta \) in order for \( A + \Delta \) to be satisfiable in \( I \).

**Definition 5.4 (Satisfiability).** Let \( I \in \mathcal{I} \), then \( \models \) is defined as follows:

1. \( I \models A[e] \) if \( e \in A \);
2. \( I \models A[B \setminus A] \) for \( B \in I \);
3. \( I \models G_1 \not\sqsubset G_2 \), \( \Delta[A] \) if \( I \models G_1, G_2, \Delta[A] \);
4. \( I \not\models G_1 \not\sqsubset G_2, \Delta[A] \bullet A_2 \) if \( I \models G_1, \Delta[A_1], I \not\models G_2, \Delta[A_2] \).

Given a finite interpretation \( I \) and a context \( \Delta \), the previous definition gives us an algorithm to compute all facts \( A \) such that \( I \models \Delta[A] \) holds. Furthermore, the relation \( \models \) satisfies the following properties.

**Lemma 5.5.** Given \( I, J \in \mathcal{I} \),

1. if \( I \models \Delta[A] \), then \( [I] \models \Delta[A'] \) for all \( A' \) s.t. \( A \preceq A' \);
2. if \( [I] \models \Delta[A] \), then there exists \( A \) such that \( I \models \Delta[A] \) and \( A \preceq A' \);
3. if \( I \models \Delta[A] \) and \( I \preceq J \), then there exists \( A' \) such that \( J \models \Delta[A'] \) and \( A' \preceq A \);
4. given a chain of abstract Herbrand interpretations \( I_1 \preceq I_2 \preceq \ldots \), if \( \bigcup_{i=1}^{\infty} [I_i] \models \Delta[A] \) then there exists \( k \) s.t. \( [I_k] \models \Delta[A] \).

The abstract fixpoint operator \( S_P : \mathcal{I} \leadsto \mathcal{I} \) 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 \( H = a_1 \not\sqsubset \ldots \not\sqsubset a_n \), then \( H \) 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:

\[
S_P(I) = \{ \check{H} + A \mid H \dashv \vdash G \in P, I \models G[A] \}
\]

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

**Proposition 5.7.** \( S_P \) is monotonic and continuous wrt. \( \preceq \).

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 \( \text{Symb}(F(P)) \) be the symbol of \( F(P) \), then we have the following main Theorem.

**Theorem 5.10 (Soundness and Completeness).** Given an LO program \( P \), \( O(P) = F(P) = \{\text{Symb}(F(P))\} \). Furthermore, there exists \( k \in N \) such that \( \text{Symb}(F(P)) = \bigcup_{i=0}^{k} S_P^i(\emptyset) \).

**Proof.** Theorem 4.7 and Corollary 5.9 show that \( O(P) = F(P) = \{\text{Symb}(F(P))\} \). 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. \( a \dashv b \not\sqsubset c \)
2. \( b \dashv (a \not\sqsubset c) & f \)
3. \( c \not\sqsubset d \dashv \top \)
4. \( c \not\sqsubset e \dashv c \not\sqsubset b \)
5. \( c \not\sqsubset f \dashv \top \)

We start the computation from \( S_P 0 = \emptyset \). The first step consists in adding the multisets corresponding to program facts, i.e., clauses 3. and 5., therefore we compute

\[
S_P^1 = \{\{c, d\}, \{c, f\}\}
\]
We name the resulting language LO. Consider an extension of LO where goal formulas range over powerful programming constructs. In this paper we will be extended in order to take into consideration more counting resources. Provability in LO amounts to provability in the proof system for LO augmented with the 1 rule. As for LO, let us define the top-down operational semantics of an LO program as follows:

\[ O_1(P) = \{ A \mid A \text{ is a fact and } P \vdash_1 A \text{ is provable} \} \]

Now the question is: is it still possible to find a finite representation of \( O_1(P) \)? We first note that, in contrast with Prop. 3.1, the weakening rule is not admissible in LO1. This implies that we cannot use the same techniques we used for the fragment without 1. More formally, the following proposition gives a negative answer to our question.

**Proposition 6.1.** Given an LO1 program \( P \), there is no algorithm to compute \( O_1(P) \).

**Proof.** To prove the result we present an encoding of Vector Addition Systems (VAS) into LO1 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 \( x'_i = x_i + \delta_i, \ldots, x'_n = x_n + \delta_n \) where \( \delta_n \) 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 LO1 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 \( H \vdash B \) where \( O c c_B(a_i) = -\delta_i \) if \( \delta_i < 0 \) (tokens removed from place \( i \)) and \( O c c_B(a_i) = \delta_i \geq 0 \) (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 \( H_i \vdash \top \) is such that if \( M_i \) is the assignment \( \langle x_1 = c_1, \ldots, x_n = c_n \rangle \) then \( O c c_B(a_j) = \delta_j \) for \( \delta_j < 0 \) and \( \delta_j \geq 0 \) for \( \delta_j > 0 \). Based on this idea, if \( P \) 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 LO1 programs as we will show in the following section. Before going into more details, we first rephrase the semantics of Section 4 for LO1. In the rest of the paper we will still denote the satisfiability judgments for LO1 with \( \models \) and \( \vdash \).

**Definition 6.2 (Satisfiability in LO1).**
Let $I \subseteq B_P$, then $\models$ is defined as follows:
1) $I \models \top, \mathcal{A}[\mathcal{A}']$ for any fact $\mathcal{A}'$;
2) $I \models \mathcal{A}[\mathcal{A}']$ if $\mathcal{A} + \mathcal{A}' \in I$;
3) $I \models G_1 \cup G_2, \Delta[\mathcal{A}]$ if $I \models G_1, G_2, \Delta[\mathcal{A}]$;
4) $I \models G_1 \cup G_2, \Delta[\mathcal{A}]$ 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 $\Delta$, and fact $\mathcal{A}$,

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

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 $T_p^1$ is defined as $T_p^1 = \{ H + \mathcal{A} \mid H \circ G \in P, I \models G[\mathcal{A}] \}$.

The following property holds.

**Proposition 6.5.** $T_p^1$ is monotonic and continuous wrt. $\subseteq$.

The fixpoint semantics is defined as $F_1(P) = \{ \mathcal{F}(T_p^1) = T_p^1 \}$, 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 LO1 program $P$, $F_1(P) = G_0(P)$.

7. **CONSTRAINT SEMANTICS FOR LO1**

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 $\mathbf{x} = (x_1, \ldots, x_n)$ 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

$$\varphi[\mathcal{A}] \equiv \bigwedge_{i=1}^{n} x_i \geq \text{Occ}_A(a_i).$$

Then all the operations on multisets involved in the definition of $S_F$ (see Def. 5.1) 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

$$\varphi[\mathcal{A} \bullet \mathcal{B}] = \varphi[\mathcal{A}] \land \varphi[\mathcal{B}],$$

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

$$\varphi[\mathcal{B} \setminus \mathcal{A}] = \exists x'. (\varphi[\mathcal{B}][x'/x] \land \rho_A(x, x')),$$

where

$$\rho_A(x, x') \equiv \bigwedge_{i=1}^{n} x_i = x'_i - \text{Occ}_A(a_i) \land x_i \geq 0.$$
its equivalence class, i.e., we will simply write \( \varphi \) instead of \( [\varphi] \).

Let \( LC_\Sigma \) be the set of (equivalence classes of) of linear constraints over the variables \( x = (x_1, \ldots, x_n) \) associated to the signature \( \Sigma = \{ a_1, \ldots, a_n \} \). The operator \( S_P^1 \) is defined on constraint interpretations consisting of sets (disjunctions) of (equivalence classes of) linear constraints. The denotation \( [I]_1 \) of a constraint interpretation \( I \) extends the one for constraints as expected: \( [I]_1 = \{ [\varphi]_1 \mid \varphi \in I \} \). 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 \( \mathcal{C} = (\mathcal{P}(LC_\Sigma), \subseteq) \) 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. \( x = 0 \land x \geq 1 \) has no solutions thus the following rules cannot be applied to this case).

Definition 7.2 (Satisfiability in \( LO_1 \)). Let \( I \in \mathcal{C} \), then \( \vdash \) is defined as follows:

\[
\begin{align*}
&I \vdash \{ \varphi \} \text{ where } \varphi \equiv x_1 = 0 \land \ldots \land x_n = 0; \\
&I \vdash \top \land A[\varphi] \text{ where } \varphi \equiv x_1 \geq 0 \land \ldots \land x_n \geq 0; \\
&I \vdash A[\varphi] \text{ where } \varphi \equiv \exists x'. (\psi[x'/x] \land \rho_A(x, x')) \in I; \\
&I \vdash G_1 \subseteq G_2, \Delta[\varphi] \text{ if } I \vdash G_1, G_2, \Delta[\varphi]; \\
&I \vdash G_1 \lor G_2, \Delta[\varphi_1 \land \varphi_2] \text{ if } I \vdash G_1, \Delta[\varphi_1], I \vdash G_2, \Delta[\varphi_2].
\end{align*}
\]

The relation \( \vdash \) satisfies the following properties.

Lemma 7.3. Given \( I, J \in \mathcal{C} \),

1. if \( I \vdash \Delta[\varphi] \), then \( [I]_1 \vdash \Delta[A] \) for every \( A \in [\varphi]_1 \);
2. if \( [I]_1 \vdash \Delta[A] \), then there exists \( \varphi \) such that \( I \vdash \Delta[\varphi] \) and \( A \in [\varphi]_1 \);
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 \( \bigcup_{i=1}^n I_i \vdash \Delta[\varphi] \) then there exists \( k \) s.t. \( I_k \vdash \Delta[\varphi] \).

We are now ready to define the extended operator \( S_P^1 \).

Definition 7.4 (Symbolic Fixpoint Operator \( S_P^1 \)). Given an \( LO_1 \) program \( P \), the operator \( S_P^1 \) is defined as follows:

\[
S_P^1(I) = \{ \varphi \mid \text{H \sim G \in P, } I \vdash G[\psi], \varphi \equiv \exists x'. (\psi[x'/x] \land \rho_H(x, x')) \}
\]

The new operator satisfies the following property.

Proposition 7.5. The operator \( S_P^1 \) is monotonic and continuous over the lattice \( \mathcal{C} \).

Furthermore, it is a symbolic version of the ground operator \( T_F^1 \), as stated below.

Proposition 7.6. Let \( I \in \mathcal{C} \), then \([S_P^1(I)]_1 = T_F^1([I]_1)\).

Corollary 7.7. \([\text{lfp}(S_P^1)]_1 = \text{lfp}(T_F^1)\).

Now, let \( \text{SymbF}_P(1) = \text{lfp}(S_P^1) \), then we have the following main theorem that shows that \( S_P^1 \) can be used (with termination guarantee) to compute symbolically the set of logical consequences of an \( LO_1 \) program.

Theorem 7.8 (Soundness and Completeness). Given an \( LO_1 \) program \( P \), \( O_1(P) = \text{F}_1(P) = \text{SymbF}_P(1) \).

8. BOTTOM-UP EVALUATION FOR \( LO_1 \)

Using a constraint-based representation for \( LO_1 \) provable multisets, we have reduced the problem of computing \( O_1(P) \) 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 \( S_P^1 \), introduced in section 7, is defined over the lattice \( \mathcal{C} = (\mathcal{P}(LC_\Sigma), \subseteq) \), with set inclusion being the partial order relation and set union the least upper bound operator. When we come to a concrete implementation of \( S_P^1 \), it is worth considering a weaker ordering relation between interpretations, namely pointwise subsumption. Let \( \preceq \) be the partial order between (equivalence classes of) constraints given by \( \varphi \preceq \psi \) if and only if \( [\varphi]_1 \subseteq [\psi]_1 \). Then we say that an interpretation \( I \) is subsumed by an interpretation \( J \), written \( I \subseteq J \), if and only if for every \( \varphi \in I \) there exists \( \psi \in J \) such that \( \psi \preceq \varphi \).

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 \( I \subseteq J \) and \( J \subseteq I \) hold. In this way, we get a lattice of interpretations ordered by \( \preceq \) 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 \( LO_1 \) programs (i.e. without the constant 1) it turns out that we need only consider constraints of the form \( x \geq c \), which can be abstracted away by considering the upward closure of \( \mathcal{C} \), as we did in Section 5. The reader can note that the \( \preceq \) relation defined above for constraints is an extension of the multiset inclusion relation we used in Section 5.

The construction based on \( \preceq \) 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 \( \preceq \) is stronger than \( \subseteq \), therefore a computation based on \( \subseteq \) is correct and it terminates.
example.

**Example 8.1.** We calculate the fixpoint semantics for the following LO1 program made up of six clauses:

1. \( a \prec 1 \)
2. \( a \bowtie b \bowrightarrow \top \)
3. \( c \bowtie c \bowrightarrow \top \)
4. \( b \bowtie b \bowrightarrow a \)
5. \( a \bowrightarrow b \)
6. \( a \bowrightarrow a \land b \)

Let \( \Sigma = \{a, b, c\} \) and consider constraints over the variables \( x = (x_a, x_b, x_c) \). We have that \( S_P = \emptyset \uplus \{x_a = 0 \land x_b = 0 \land x_c = 0\} \), therefore, by the first clause, \( \varphi \in S_P \bowtie 1 \), where \( \varphi \equiv 3x'.(x'_a = 0 \land x'_b = 0 \land x'_c = x'_a + 1 \land x_c = x'_c \land x_c = x'_c) \), which is equivalent to \( x = 1 \land x_b = 0 \land x_c = 0 \).

Now from now on, we leave to the reader the details concerning the equivalence of constraints. By reasoning in a similar way, using clauses 2. and 3. we calculate \( S_P \bowtie 1 \) (see Fig. 3).

We now compute \( S_P \bowtie 2 \). By 4., as \( S_P \bowtie 1 \uplus \{x_a = 0 \land x_b \geq 0 \land x_c = 0 \in S_P \bowtie 1 \} \) we get \( x_a = 0 \land x_b \geq 2 \land x_c = 0 \), and, similarly, we get \( x_a \geq 0 \land x_b \geq 3 \land x_c \geq 0 \). By 5., we have \( x_a \geq 2 \land x_b \geq 0 \land x_c \geq 0 \), while clause 6. is not (yet) applicable.

Therefore, modulo redundant constraints (i.e. constraints subsumed by the already calculated ones) the value of \( S_P \bowtie 2 \) is given in Fig. 3.

Now, we can compute \( S_P \bowtie 3 \). By 4. and \( x_a \geq 2 \land x_b \geq 0 \land x_c \geq 0 \in S_P \bowtie 1 \) we get \( x_a \geq 1 \land x_b \geq 2 \land x_c \geq 0 \), which is subsumed by \( x_a \geq 1 \land x_b \geq 1 \land x_c \geq 0 \). By 5. and \( x_a = 0 \land x_b \geq 2 \land x_c = 0 \), we get \( x_a \geq 1 \land x_b \geq 1 \land x_c = 0 \), subsumed by \( x_a \geq 1 \land x_b \geq 1 \land x_c \geq 0 \). Similarly, by 5. and \( x_a \geq 0 \land x_b \geq 3 \land x_c \geq 0 \) we get redundant information. By 6., from \( x_a \geq 1 \land x_b \geq 1 \land x_c \geq 0 \) and \( x_a = 0 \land x_b = 2 \land x_c = 0 \) we get \( x_a = 0 \land x_b = 1 \land x_c = 1 \), from \( x_a \geq 1 \land x_b \geq 1 \land x_c \geq 0 \) and \( x_a \geq 0 \land x_b \geq 3 \land x_c \geq 0 \) we get \( x_a \geq 0 \land x_b \geq 2 \land x_c \geq 1 \), and finally from \( x_a \geq 2 \land x_b \geq 0 \land x_c \geq 0 \) and \( x_a \geq 1 \land x_b \geq 1 \land x_c \geq 0 \) we have \( x_a \geq 1 \land x_b \geq 0 \land x_c \geq 1 \). 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 \( S_P \bowtie 1 \). We have then

\( S_P \bowtie 1 = S_P \bowtie 2 \bowtie 3 = \text{Sym}bF_1(P) \), so that in this particular case we can reformulate the operation of the semantic operators of P using the more suggestive multiset notation (we recall that \( [A] = \{B \mid A \bowright B\} \), where \( \bowright \) is multiset inclusion):

\[
O_2(P) = F_1(P) = \{a, b, \{b, c\}, \emptyset\} \cup \{a, b, \{c, c\}, \{b, b\}, \{a, a\}, \{b, b, c\}, \{a, c\}\}
\]

\( \square \)

It is often not the case that the symbolic computation of LO1 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 LO1 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 \( S_P \).

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 \( S_P^\uparrow \) 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 \( \models \)) to handle formulas with the connectives \( \land, \top \) and \( \bot \).

Other sources of inspiration come from linear logic programming. In [18], Harland and Winkoff 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 LO1, 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 \( \bot \)). 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 \( B \), that has to be proved is subsumed by a node already visited, say \( A \), (i.e. \( B = A + A' \)), then the part of proof tree between the two goals is replaced by a proof tree for \( A + (A')' \). 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 \( \bowright \)). Finally, differently from [6], in this paper we have given also a formal semantics for the extension of LO with the constant \( \bot \).

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
Linear Logic Programming.

To conclude, let us discuss the directions of research related to the connection we establish in this paper. The 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


