Theory and Practice of Logic Programming **2** (1): 85–122, January 2002.

Printed in the United Kingdom © 2002 Cambridge University Press

An effective fixpoint semantics for linear logic programs

MARCO BOZZANO, GIORGIO DELZANNO and MAURIZIO MARTELLI

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

Abstract

In this paper we investigate the theoretical foundation of a new bottom-up semantics for linear logic programs, and more precisely for the fragment of LinLog (Andreoli, 1992) that consists of the language LO (Andreoli & Pareschi, 1991) enriched with the constant **1**. We use *constraints* to symbolically and finitely represent possibly infinite collections of provable goals. We define a fixpoint semantics based on a new operator in the style of T_P working over constraints. An application of the fixpoint operator can be computed algorithmically. As sufficient conditions for termination, we show that the fixpoint computation is guaranteed to converge for propositional LO. To our knowledge, this is the first attempt to define an effective fixpoint semantics for linear logic programs. As an application of our framework, we also present a formal investigation of the relations between LO and Disjunctive Logic Programming (Minker *et al.*, 1991). Using an approach based on abstract interpretation, we show that DLP fixpoint semantics can be viewed as an abstraction of our semantics for LO. We prove that the resulting abstraction is *correct* and *complete* (Cousot & Cousot, 1977; Giacobazzi & Ranzato, 1997) for an interesting class of LO programs encoding Petri Nets.

KEYWORDS: program analysis, Boolean functions

1 Introduction

In recent years a number of fragments of linear logic (Girard, 1987) have been proposed as a logical foundation for extensions of logic programming (Miller, 1995). Several new programming languages like LO (Andreoli & Pareschi, 1991), LinLog (Andreoli, 1992), ACL (Kobayashi & Yonezawa, 1995), Lolli (Hodas & Miller, 1994) and Lygon (Harland & Pym, 1994) 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 (1992) and the *uniform* proofs of Miller (1996) 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 the fragment

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

of linear logic underlying the language LO (Andreoli & Pareschi, 1991), and its proper extension with the constant **1**. Both languages can be seen as fragments of LinLog (Andreoli, 1992), which is a presentation of full linear logic. Throughout the paper, we will simply refer to these two fragments as LO and $LO_{\mathbf{1}}$. The reason we selected these fragments is that we were looking for a relatively simple linear logic language with a uniform-proof presentation, state-based computations and aspects of concurrency. Considering both LO and its extension with the constant **1** will help us to formally classify the different expressive power of linear logic connectives like $\parr$, $\&$, $\top$ and **1** when incorporated into a logic programming setting. In practice, LO has been successfully applied to model concurrent object-oriented languages (Andreoli & Pareschi, 1991), and multi-agent coordination languages based on the Linda model (Andreoli, 1996).

The operational semantics we propose consists of a *goal-independent bottom-up* evaluation of 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 Harland & Winikoff (1998), 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. Recent results connecting verification techniques and semantics of logic programs (Delzanno & Podelski, 1999) show that bottom-up evaluation can be used to automatically check *properties* (specified in temporal logic like CTL) of the original program. In this paper will go further showing that the *provability* relation in logic programming languages like LO can be used to naturally express verification problems for Petri Nets-like models of concurrent systems. Finally, a formal definition of the bottom-up semantics can be useful for studying equivalence, compositionality and abstract interpretation, as for traditional logic programs (Bossi *et al*., 1994; Gabbrielli *et al*., 1995).

Technically, our contributions are as follows. We first consider a formulation of LO with $\parr$, $\multimap$, $\&$ and $\top$. Following the semantic framework of (constraint) logic programming (Gabbrielli *et al*., 1995; Jaffar & Maher, 1994), 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 structure of the LO proof system, already in the propositional case there are infinitely many provable multisets of atomic formulas. 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* (w.r.t. multiset inclusion) provable multisets. Dickson's Lemma (Dickson, 1913) 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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

for model checking of infinite-state systems given in Abdulla *et al.* (1996) and Finkel & Schnoebelen (2001).

The decidability of propositional provability shows that LO is not as interesting as one could expect from a state-oriented extension of the logic programming paradigm. Specifically, LO does not provide a natural way to *count* resources. This feature can be introduced by a slight extension of LO in which we add unit clauses defined via the constant **1**. The resulting language, namely $\mathrm{LO}_{\mathbf{1}}$, can be viewed as a first step towards more complex languages based on linear logic like LinLog (Andreoli, 1992). As we show in this paper, $\mathrm{LO}_{\mathbf{1}}$ allows to model more sophisticated models of concurrent systems than LO, e.g. in $\mathrm{LO}_{\mathbf{1}}$ it is possible to model Petri Nets with *transfer arcs*. Adding the constant **1** breaks down the decidability of provability in propositional LO. Despite this negative result, it is still possible to define an effective $S_P^{\mathbf{1}}$ operator for $\mathrm{LO}_{\mathbf{1}}$. For this purpose, as *symbolic* representation of potentially infinite sets of contexts, we choose a special class of *linear constraints* defined over variables that *count* resources. This abstract domain generalizes the domain used for LO: the latter can be represented as the subclass of constraints with 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. see Bultan *et al.* (1997), Delzanno & Podelski (1999) and Henzinger *et al.* (1997)) and abstract interpretation (Cousot & Halbwachs, 1978) 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 Andreoli *et al.* (1997), 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 linear logic programs.

Our semantic framework can also be used as a tool to compare the relative strength of different logic programming extensions. As an application, we shall present a detailed comparison between LO and Disjunctive Logic Programming (DLP). Though DLP has been introduced in order to represent 'uncertain' beliefs, a closer look at its formal definition reveals very interesting connections with the paradigm of linear logic programming: both DLP and LO programs extend Horn programs allowing clauses with *multiple* heads. In fact, in DLP we find clauses of the form

$$p(X) \vee q(X) \leftarrow r(X) \wedge t(X),$$

whereas in LO we find clauses of the form

$$p(X) \parr q(X) \circ\!\!- r(X) \,\&\, t(X).$$

To understand the differences, we must look at the operational semantics of DLP programs. In DLP, a resolution step is extended so as to work over positive clauses (*sets/disjunctions* of facts). Implicit *contraction* steps are applied over the selected clause. On the contrary, being in a sub-structural logic in which contraction is forbidden, we know that LO resolution behaves as *multiset* rewriting. Following the bottom-up approach that we pursue in this paper, we will exploit the classical framework of abstract interpretation to formally compare the two languages. Technically,

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

we first specialize our fixpoint semantics to a *flat* fragment of propositional LO (i.e. arbitrary nesting of connectives in goals is forbidden) which directly corresponds to DLP as defined in Minker *et al.* (1991). Then, by using an abstract-interpretation-based approach, we exhibit a Galois connection between the semantic domains of DLP and LO, and we show that the semantics of DLP programs can be described as an abstraction of the semantics of LO programs. Using the theory of abstract interpretation and the concept of *complete abstraction* (Cousot & Cousot, 1977; Giacobazzi & Ranzato, 1997) we discuss the quality of the resulting abstraction. This view of DLP as an abstraction of LO is appealing for several reasons. First of all, it opens the possibility of using techniques developed for DLP for the analysis of LO programs. Furthermore, it shows that the paradigm of DLP could have unexpected applications as a framework to reason about properties of Petri Nets, a well-know formalism for concurrent computations (Karp & Miller, 1969). In fact, as we will prove formally in the paper, DLP represents a *complete* abstract domain for LO programs that encode Petri Nets.

Plan of the paper

After introducing some notations in Section 2, in Section 3 we recall the main features of LO (Andreoli & Pareschi, 1991). In Section 4 we introduce the so-called *ground* semantics, via the T_P operator, and prove that the least fixpoint of T_P characterizes the operational semantics of an LO program. In Section 5 we reformulate LO semantics by means of the *symbolic* S_P operator, and we relate it to T_P. In Section 6 we consider an extended fragment of LO with the constant **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. As an application of our framework, in Sections 9 and 10 we investigate the relations between LO and DLP, and in Section 11 we investigate the relations with Petri Nets. Finally, in Sections 12 and 13 we discuss related works and conclusions.

This paper is an extended version of earlier work (Bozzano *et al.*, 2000a, 2000b).

2 Preliminaries

In this paper, we will extensively use operations on multisets. We will consider a fixed signature, i.e. a finite set of propositional symbols, $\Sigma = \{a_1, \ldots, a_n\}$. Multisets over Σ will be hereafter called *facts*, and symbolically noted as $\mathscr{A}, \mathscr{B}, \mathscr{C}, \ldots$. A multiset with (possibly duplicated) elements $b_1, \ldots, b_m \in \Sigma$ will be simply indicated as $\{b_1, \ldots, b_m\}$, overloading the usual notation for sets.

A fact $\mathscr{A}$ is uniquely determined by a finite map $Occ : \Sigma \to \mathrm{N}$ such that $Occ_{\mathscr{A}}(a_i)$ is the number of occurrences of a_i in $\mathscr{A}$. Facts are ordered according to the *multiset inclusion* relation $\preccurlyeq$ defined as follows: $\mathscr{A} \preccurlyeq \mathscr{B}$ if and only if $Occ_{\mathscr{A}}(a_i) \leqslant Occ_{\mathscr{B}}(a_i)$ 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 $\epsilon \preccurlyeq \mathscr{A}$ for any $\mathscr{A}$. The *multiset union* $\mathscr{A}, \mathscr{B}$ (alternatively $\mathscr{A} + \mathscr{B}$ when ',' is ambiguous) of two facts $\mathscr{A}$ and $\mathscr{B}$ is such that

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

$Occ_{\mathscr{A},\mathscr{B}}(a_i) = Occ_{\mathscr{A}}(a_i) + Occ_{\mathscr{B}}(a_i)$ for $i : 1,\ldots,n$. The *multiset difference* $\mathscr{A} \setminus \mathscr{B}$ is such that $Occ_{\mathscr{A}\setminus\mathscr{B}}(a_i) = max(0, Occ_{\mathscr{A}}(a_i) - Occ_{\mathscr{B}}(a_i))$ for $i : 1,\ldots,n$. We define a special operation $\bullet$ to compute the *least upper bound* of two facts with respect to $\preccurlyeq$. Namely, $\mathscr{A} \bullet \mathscr{B}$ is such that $Occ_{\mathscr{A}\bullet\mathscr{B}}(a_i) = max(Occ_{\mathscr{A}}(a_i), Occ_{\mathscr{B}}(a_i))$ for $i : 1,\ldots,n$. Finally, we will use the notation $\mathscr{A}^n$, where n is a natural number, to indicate $\mathscr{A} + \ldots + \mathscr{A}$ (n times).

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 $H = a_1 \parr \ldots \parr a_n$, we introduce the notation $\widehat{H}$ to denote the multiset $a_1,\ldots,a_n$.

Finally, let $T : \mathscr{I} \to \mathscr{I}$ be an operator defined over a complete lattice $\langle \mathscr{I}, \sqsubseteq \rangle$. We define $T\uparrow_0 = \emptyset$, where $\emptyset$ is the bottom element, $T\uparrow_{k+1} = T(T\uparrow_k)$ for all $k \geqslant 0$, and $T\uparrow_\omega = \bigsqcup_{k=0}^{\infty} T\uparrow_k$, where $\bigsqcup$ is the least upper bound w.r.t. $\sqsubseteq$. Furthermore, we use $lfp(T)$ to denote the *least fixpoint* of T.

3 The programming language LO

LO (Andreoli & Pareschi, 1991) 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 $\multimap$ (*linear implication*), & (*additive conjunction*), $\parr$ (*multiplicative disjunction*), and the constant $\top$ (*additive identity*). In the propositional case LO consists of the following class of formulas:

$$\mathbf{D} ::= \mathbf{A}_1 \parr \ldots \parr \mathbf{A}_n \multimap \mathbf{G} \mid \mathbf{D} \,\&\, \mathbf{D}$$

$$\mathbf{G} ::= \mathbf{G} \parr \mathbf{G} \mid \mathbf{G} \,\&\, \mathbf{G} \mid \mathbf{A} \mid \top$$

Here $\mathbf{A}_1,\ldots,\mathbf{A}_n$ and $\mathbf{A}$ range over propositional symbols from a fixed signature Σ. **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:

$$!C_1,\ldots,!C_n \;\rightarrow\; G_1,\ldots,G_k.$$

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. 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 Figure 1. In Figure 1, P is a set of implicational clauses, $\mathscr{A}$ denotes a multiset of atomic formulas, whereas Δ denotes a multiset of **G**-formulas. A sequent is provable if all branches of its proof tree terminate with instances of the $\top_r$ axiom. The proof system of

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

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

$$\frac{P \Rightarrow G, \mathscr{A}}{P \Rightarrow \widehat{H} + \mathscr{A}}\ bc \quad (H \multimap G \in P)$$

Fig. 1. A proof system for LO.

Figure 1 is a specialization of more general uniform proof systems for linear logic like Andreoli's focusing proofs (Andreoli, 1992), and Forum (Miller, 1996).

The rule *bc* denotes a backchaining (resolution) step ($\widehat{H}$ is the multiset consisting of the literals in the disjunction H, see Section 2). 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

$$a_1 \parr \ldots \parr a_n \multimap \top$$

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 $\mathscr{A}$, as shown in the following scheme:

$$\frac{\dfrac{}{P \Rightarrow \top, \mathscr{A}}\ \top_r}{P \Rightarrow a_1, \ldots, a_n, \mathscr{A}}\ bc$$

$$provided \quad a_1 \parr \ldots \parr a_n \multimap \top \in P$$

This observation leads us to the following property (we recall that $\preccurlyeq$ is the sub-multiset relation).

Proposition 1
Given an LO program P and two contexts Δ, Δ' such that $\Delta \preccurlyeq \Delta'$, if $P \Rightarrow \Delta$ then $P \Rightarrow \Delta'$.

Proof
By simple induction on the structure of LO proofs. □

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. Thus, LO can be viewed as an *affine* fragment of linear logic. Note that weakening and contraction are both admissible on the left hand side (i.e. on the program part) of LO sequents.

Example 1
Let P be the LO program consisting of the clauses

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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \Rightarrow e, \top}\,\top_r}{P \Rightarrow d, e, c}\,bc^{(3)}}{P \Rightarrow d \parr e, c}\,\parr_r \qquad \dfrac{\dfrac{}{P \Rightarrow \top}\,\top_r}{P \Rightarrow f, c}\,bc^{(5)}}{P \Rightarrow (d \parr e) \,\&\, f, c}\,\&_r}{P \Rightarrow b, c}\,bc^{(2)}}{P \Rightarrow b \parr c}\,\parr_r}{P \Rightarrow e, e}\,bc^{(4)}$$

Fig. 2. An LO proof for the goal e, e in the program of Example 1.

and consider an initial goal e, e. A proof for this goal is shown in Figure 2, where we have denoted by $bc^{(i)}$ the application of the backchaining rule over clause number i of P. The proof proceeds as follows. Using clause 4, to prove e, e we have to prove $b \parr c$, which, by LO $\parr_r$ rule, reduces to prove b, c. At this point we can backchain over clause 2, and we get the new goal $(d \parr e) \,\&\, f, c$. By applying $\&_r$ rule, we get two separate goals $d \parr e, c$ and f, c. The first, after a reduction via $\parr_r$ rule, is provable by means of clause (axiom) 3, while the latter is provable directly by clause (axiom) 5. Note that $\top$ succeeds in a non-empty context (i.e. containing e) in the left branch. A similar proof shows that the goal a is also provable from P. By Proposition 1, provability of e, e and a implies provability of any multiset of goals e, e, Δ and a, Δ, for every context Δ. □

We conclude this section with the definition of the following induction measure on LO goals, which we will later need in proofs.

Definition 3.1
Given a goal G, the induction measure $m(G)$ is defined according to the following rules: $m(A) = 0$ for every atomic formula A; $m(\top) = 0$; $m(G_1 \,\&\, G_2) = m(G_1 \parr G_2) = m(G_1) + m(G_2) + 1$. The induction measure extends to contexts by defining $m(G_1, \ldots, G_n) = m(G_1) + \ldots + m(G_n)$.

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. Formally, we define the operational *top-down* semantics of an LO program P as follows:

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

Note that the information on provable *facts* from a given program P is all we need to decide whether a general goal (with possible nesting of connectives) is provable from P or not. This is a consequence of the *focusing* property (Andreoli, 1992) of

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

LO provability, which ensures that provability of a compound goal can always be reduced to provability of a finite set of atomic multisets. In a similar way, in Prolog the standard bottom-up semantics is defined as a set of *atoms*, while in general *conjunctions* of atoms are allowed in clause bodies.

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 define a procedure to compute all goal formulas G such that G is provable from P. This procedure should enjoy the usual properties of classical bottom-up semantics, in particular its definition should be based on an *effective* fixpoint operator (i.e. at least every single step must be finitely computable), and it should be *goal-independent*. As usual, goal independence is achieved by searching for proofs starting from the axioms (the unit clauses of Section 3) and accumulating goals which can be proved by applying program clauses to the current interpretation. As for the operational semantics, we can limit ourselves to goal formulas consisting of multisets of atomic formulas, without any loss of generality. 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, the Herbrand base of P, denoted B_P, is given by

$$B_P = \{\mathscr{A} \mid \mathscr{A} \text{ is a multiset (fact) over } \Sigma\}.$$

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

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 $I \models \Delta[\mathscr{A}]$. The need for this judgment, with respect to the familiar logic programming setting (Gabbrielli *et al.*, 1995), is motivated by the arbitrary nesting of connectives in LO clause bodies, which is not allowed in traditional presentations of (constraint) logic programs. In $I \models \Delta[\mathscr{A}]$, $\mathscr{A}$ should be read as an *output* fact such that $\mathscr{A} + \Delta$ is valid in I. This notion of *satisfiability* is modeled according to the right-introduction rules of the connectives. The notion of output fact $\mathscr{A}$ will simplify the presentation of the algorithmic version of the judgment which we will present in Section 5.

Definition 4.3 (*Satisfiability*)
Let I be a Herbrand interpretation, then $\models$ is defined as follows:

$$I \models \top, \Delta[\mathscr{A}'] \text{ for any fact } \mathscr{A}';$$

$$I \models \mathscr{A}[\mathscr{A}'] \text{ if } \mathscr{A} + \mathscr{A}' \in I;$$

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

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

The relation $\models$ satisfies the following properties.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Lemma 1
For any interpretations I, J, context Δ, and fact $\mathscr{A}$,

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

Proof
The proof of (i) and (ii) is by simple induction. The proof of (iii) is by (complete) induction on $m(\Delta)$ (see Definition 3.1).

- If $\Delta = \top, \Delta'$, then, no matter which k you choose, $I_k \models \top, \Delta'[\mathscr{A}]$;
- if Δ is a fact, then $\bigcup_{i=1}^{\infty} I_i \models \Delta[\mathscr{A}]$ means $\Delta, \mathscr{A} \in \bigcup_{i=1}^{\infty} I_i$, which in turn implies that there exists k such that $\Delta, \mathscr{A} \in I_k$, therefore $I_k \models \Delta[\mathscr{A}]$;
- if $\Delta = G_1 \,\&\, G_2, \Delta'$, then by inductive hypothesis there exist k_1 and k_2 s.t. $I_{k_1} \models G_1, \Delta'[\mathscr{A}]$ and $I_{k_2} \models G_2, \Delta'[\mathscr{A}]$. Therefore, if $k = max\{k_1, k_2\}$, by (ii) we have that $I_k \models G_1, \Delta'[\mathscr{A}]$ and $I_k \models G_2, \Delta'[\mathscr{A}]$, therefore $I_k \models G_1 \,\&\, G_2, \Delta'[\mathscr{A}]$, i.e. $I_k \models \Delta[\mathscr{A}]$ as required;
- the $\parr$-case follows by a straightforward application of the inductive hypothesis.

□

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

Definition 4.4 (*Fixpoint operator* T_P)
Given a program P and an interpretation I, the operator T_P is defined as follows:

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

The following property holds.

Proposition 2
For every program P, T_p is monotonic and continuous over the lattice $\langle \mathscr{D}, \subseteq \rangle$.

Proof
Monotonicity. Immediate from T_P definition and Lemma 1 (ii).

Continuity. We prove that T_P is finitary. Namely, given an increasing chain of interpretations $I_1 \subseteq I_2 \subseteq \ldots$, T_P is finitary if $T_P(\bigcup_{i=1}^{\infty} I_i) \subseteq \bigcup_{i=1}^{\infty} T_P(I_i)$. We simply need to show that if $T_P(\bigcup_{i=1}^{\infty} I_i) \models \Delta[\epsilon]$ then there exists k such that $T_P(I_k) \models \Delta[\epsilon]$). The proof is by induction on $m(\Delta)$.

- If $\Delta = \top, \Delta'$, then, no matter which k you choose, $T_P(I_k) \models \top, \Delta'[\epsilon]$;
- if Δ is a fact and $\Delta \in T_P(\bigcup_{i=1}^{\infty} I_i)$, then, by definition of T_P, there exist a fact $\mathscr{A}$ and a clause $A_1 \parr \ldots \parr A_n \multimap G \in P$, such that $\bigcup_{i=1}^{\infty} I_i \models G[\mathscr{A}]$ and $\Delta = A_1, \ldots, A_n, \mathscr{A}$. Lemma 1 *iii*) implies that $\exists k. I_k \models G[\mathscr{A}]$, therefore, again by definition of T_P, $T_P(I_k) \models A_1, \ldots, A_n, \mathscr{A}[\epsilon]$, i.e. $T_P(I_k) \models \Delta[\epsilon]$ as required;
- if $\Delta = G_1 \,\&\, G_2, \Delta'$, then by inductive hypothesis, there exist k_1 and k_2 s.t. $T_P(I_{k_1}) \models G_1, \Delta'[\epsilon]$ and $T_P(I_{k_2}) \models G_2, \Delta'[\epsilon]$. Then, if $k = max\{k_1, k_2\}$, by Lemma 1 *ii*) we have that $T_P(I_k) \models G_1, \Delta'[\epsilon]$ and $T_P(I_k) \models G_2, \Delta'[\epsilon]$. This implies $T_P(I_k) \models G_1 \,\&\, G_2, \Delta'[\epsilon]$, i.e. $T_P(I_k) \models \Delta[\epsilon]$ as required;

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

– the ⅋-case follows by a straightforward application of the inductive hypothesis.

□

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

Following Lloyd (1987), 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 $\{A \circ\!\!- \top \mid \widehat{A} \in I\}$, the definition of T_P can be viewed as the following instance of the *cut* rule of linear logic:

$$\frac{!P, G \to H \quad !P_I \to G, \mathscr{A}}{!P, !P_I \to H, \mathscr{A}} \; cut$$

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

$$\frac{P \Rightarrow H \circ\!\!- G \quad P_I \Rightarrow G, \mathscr{A}}{P \cup P_I \Rightarrow H, \mathscr{A}} \; cut$$

Note that, since $H \circ\!\!- G \in P$, the sequent $P \Rightarrow H \circ\!\!- G$ 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 1 (*Soundness and Completeness*)
For every LO program P, $F(P) = O(P)$.

Proof

(i) $F(P) \subseteq O(P)$**.** We prove that for every k and context Δ, if $T_P{\uparrow}_k \models \Delta[\epsilon]$ then $P \Rightarrow \Delta$. The proof is by (complete) induction on $\bar{m}(T_P{\uparrow}_k, \Delta)$, where $\bar{m}$ is an induction measure defined by $\bar{m}(T_P{\uparrow}_k, \Delta) = \langle k, m(\Delta)\rangle$, and $\langle k, m\rangle < \langle k', m'\rangle$ if and only if $(k < k')$ or $(k = k'$ and $m < m')$ (lexicographic ordering).

– If $\Delta = \top, \Delta'$, the conclusion is immediate;
– if Δ is a fact, then $\Delta \in T_P{\uparrow}_k$, so that $T_P{\uparrow}_k \neq \emptyset$ and $k > 0$. By definition of T_P we have that there exist a fact $\mathscr{A}$ and a clause $A_1 \parr \ldots \parr A_n \circ\!\!- G \in P$, such that $T_P{\uparrow}_{k-1} \models G[\mathscr{A}]$ and $\Delta = A_1, \ldots, A_n, \mathscr{A}$. By Lemma 1 *i*) we have that $T_P{\uparrow}_{k-1} \models G, \mathscr{A}[\epsilon]$, and then, by inductive hypothesis, $P \Rightarrow G, \mathscr{A}$, therefore by LO *bc* rule, $P \Rightarrow A_1, \ldots, A_n, \mathscr{A}$, i.e., $P \Rightarrow \Delta$;
– if $\Delta = G_1 \,\&\, G_2, \Delta'$ then by inductive hypothesis $P \Rightarrow G_1, \Delta'$ and $P \Rightarrow G_2, \Delta'$, therefore $P \Rightarrow G_1 \,\&\, G_2, \Delta'$ by LO $\&_r$ rule;
– the ⅋-case follows by a straightforward application of the inductive hypothesis.

(ii) $O(P) \subseteq F(P)$**.** We prove that for every context Δ if $P \Rightarrow \Delta$ then there exists k such that $T_P{\uparrow}_k \models \Delta[\epsilon]$ by induction on the structure of the LO proof.

– If the proof ends with an application of $\top_r$, then the conclusion is immediate;
– if the proof ends with an application of the *bc* rule, then $\Delta = A_1, \ldots, A_n, \mathscr{A}$, where $A_1, \ldots, A_n$ are atomic formulas, and there exists a clause $A_1 \parr \ldots \parr$

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

$A_n \multimap G \in P$. For the uniformity of LO proofs, we can suppose $\mathscr{A}$ to be a fact. By inductive hypothesis, we have that there exists k such that $T_P\uparrow_k \models G, \mathscr{A}[\epsilon]$, then, by Lemma 1 *i*), $T_P\uparrow_k \models G[\mathscr{A}]$, which, by definition of T_P, in turn implies that $A_1, \ldots, A_n, \mathscr{A} \in T_P(T_P\uparrow_k) = T_P\uparrow_{k+1}$, therefore $T_P\uparrow_{k+1} \models A_1, \ldots, A_n, \mathscr{A}[\epsilon]$, i.e. $T_P\uparrow_{k+1} \models \Delta[\epsilon]$;

– if the proof ends with an application of the $\&_r$ rule, then $\Delta = G_1 \,\&\, G_2, \Delta'$ and, by inductive hypothesis, there exist k_1 and k_2 such that $T_P\uparrow_{k_1} \models G_1, \Delta'[\epsilon]$ and $T_P\uparrow_{k_2} \models G_2, \Delta'[\epsilon]$. Then, if $k = max\{k_1, k_2\}$ we have, by Lemma 1 *ii*), that $T_P\uparrow_k \models G_1, \Delta'[\epsilon]$ and $T_P\uparrow_k \models G_2, \Delta'[\epsilon]$, therefore $T_P\uparrow_k \models G_1 \,\&\, G_2, \Delta'[\epsilon]$, i.e. $T_P\uparrow_k \models \Delta[\epsilon]$;

– the $\parr$-case follows by a straightforward application of the inductive hypothesis.

□

We note that it is also possible to define a *model-theoretic* semantics (as for classical logic programming (Gabbrielli *et al.*, 1995)) based on the notion of *least model* with respect to a given class of models and partial order relation. In our setting, 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 $H \multimap G \in P$ and for every fact $\mathscr{A}$,

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

It turns out that the operational, fixpoint and model-theoretic semantics are all equivalent. We omit details. Finally, we also note that these semantics can be proved equivalent to the *phase semantics* for LO given in Andreoli & Pareschi (1991).

5 An effective semantics for LO

The operator T_P defined in the previous section does not enjoy one of the crucial properties we required for our bottom-up semantics, namely its definition is *not* effective. As an example, take the program P consisting of the clause $a \multimap \top$. Then, $T_P(\emptyset)$ is the set of all multisets with *at least* one occurrence of a, which is an *infinite* set. In other words, $T_P(\emptyset) = \{\mathscr{B} \mid a \preccurlyeq \mathscr{B}\}$, where $\preccurlyeq$ is the multiset inclusion relation of Section 2. 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 (Abdulla *et al.*, 1996), a *constraint system*). The previous example suggests us that a provable fact $\mathscr{A}$ may be used to *implicitly* represent the ideal generated by $\mathscr{A}$, i.e. the subset of B_P defined as follows:

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

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

Definition 5.1 (*Abstract Herbrand Interpretation*)
The lattice $\langle \mathscr{I}, \sqsubseteq \rangle$ of abstract Herbrand interpretations is defined as follows:

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

- $\mathscr{I} = \mathscr{P}(B_P)/\simeq$ where $I \simeq J$ if and only if $[\![I]\!] = [\![J]\!]$;
- $[I]_\simeq \sqsubseteq [J]_\simeq$ if and only if for all $\mathscr{B} \in I$ there exists $\mathscr{A} \in J$ such that $\mathscr{A} \preccurlyeq \mathscr{B}$;
- the bottom element is the empty set $\emptyset$, the top element is the $\simeq$-equivalence class of the singleton $\{\epsilon\}$ (ϵ=empty multiset, $\epsilon \preccurlyeq \mathscr{A}$ for any $\mathscr{A} \in B_P$);
- 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 $\mathscr{A}$ is redundant in $\{\epsilon, \mathscr{A}\}$, which, in fact, is equivalent to $\{\epsilon\}$. It is important to note that to compare two ideals we simply need to compare their generators w.r.t. 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 $\mathscr{A} \preccurlyeq \mathscr{B}$, then $[\![\mathscr{B}]\!] \subseteq [\![\mathscr{A}]\!]$. 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* (Abdulla *et al.*, 1996; Finkel & Schnoebelen, 2001), as stated in Proposition 3 and Corollary 1 below. This property is the key point of our idea. In fact, it will allow us to prove that the computation of the least fixpoint of the *symbolic* formulation of the operator T_P (working on abstract Herbrand interpretations) is guaranteed to terminate on every input LO program.

Proposition 3 (*Dickson's Lemma (Dickson, 1913)*)
Let $A_1 A_2 \ldots$ 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 Abdulla *et al.* (1996), by definition of $\sqsubseteq$ the following Corollary holds.

Corollary 1
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 1 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* w.r.t. the ground semantics. If we find it, then we can use the corollary to prove that (for any program) its fixpoint can be reached after finitely many steps. For this purpose and using the multiset operations $\setminus$ (difference), $\bullet$ (least upper bound w.r.t. $\preccurlyeq$), and ϵ (empty multiset) defined in Section 2, we first define a new version of the satisfiability relation $\models$. The intuition under the new judgment $I \Vdash \Delta[\mathscr{A}]$ is that $\mathscr{A}$ is the *minimal* fact (w.r.t. multiset inclusion) that should be added to Δ in order for $\mathscr{A} + \Delta$ to be satisfiable in I.

Definition 5.2 (*Satisfiability*)
Let $I \in \mathscr{P}(B_P)$, then $\Vdash$ is defined as follows:

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

$I \Vdash \top, \Delta[\epsilon]$;

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

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

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

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

Example 2
Let us consider clause 2. of Example 1, namely

$$b \multimap (d \parr e) \,\&\, f,$$

and $I = \{\{c,d\},\{c,f\}\}$. We want to compute the facts $\mathscr{A}$ for which $I \Vdash G[\mathscr{A}]$, where $G = (d \parr e) \,\&\, f$ is the body of the clause. From the second rule defining the judgment $\Vdash$, we have that $I \Vdash \{d,e\}[\{c\}]$, because $\{c,d\} \in I$ and $\{c,d\} \setminus \{d,e\} = \{c\}$. Therefore we get $I \Vdash d \parr e[\{c\}]$ using the third rule for $\Vdash$. Similarly, we have that $I \Vdash \{f\}[\{c\}]$, because $\{c,f\} \in I$ and $\{c,f\} \setminus \{f\} = \{c\}$. By applying the fourth rule for $\Vdash$, with $G_1 = d \parr e$, $G_2 = f$, $\mathscr{A}_1 = \{c\}$, $\mathscr{A}_2 = \{c\}$ and $\Delta = \epsilon$, we get $I \Vdash G[\{c\}]$, in fact $\{c\} \bullet \{c\} = \{c\}$. There are other ways to apply the rules for $\Vdash$. In fact, we can get $I \Vdash \{d,e\}[\{c,f\}]$, because $\{c,f\} \in I$ and $\{c,f\} \setminus \{d,e\} = \{c,f\}$. Similarly, we can get $I \Vdash \{f\}[\{c,d\}]$. By considering all combinations, it turns out that $I \Vdash G[\mathscr{A}]$, for every $\mathscr{A} \in \{\{c\},\{c,f\},\{c,d\},\{c,d,f\}\}$. The information conveyed by $\{c,f\},\{c,d\},\{c,d,f\}$ is in some sense *redundant*, as we shall see in the following (see Example 3). In other words, it is not always true that the output fact of the judgment $\Vdash$ is *minimal* (in the previous example only the output $\{c\}$ is minimal). Nevertheless, the important point to be stressed here is that the set of possible facts satisfying the judgment, given I and G, is *finite*. This will be sufficient to ensure effectiveness of the fixpoint operator. □

The relation $\Vdash$ satisfies the following properties.

Lemma 2
For every $I, J \in \mathscr{P}(B_P)$, context Δ, and fact $\mathscr{A}$,

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

Proof
(i) By induction on Δ.

– $I \Vdash \top, \Delta[\epsilon]$ and $[\![I]\!] \models \top, \Delta[\mathscr{A}']$ and $\epsilon \preccurlyeq \mathscr{A}'$ for any $\mathscr{A}'$;
– if $I \Vdash \mathscr{A}[\mathscr{A}']$ then $\mathscr{A}' = \mathscr{B} \setminus \mathscr{A}$ for $\mathscr{B} \in I$. Since $\mathscr{B} \preccurlyeq (\mathscr{B} \setminus \mathscr{A}) + \mathscr{A} = \mathscr{A}' + \mathscr{A}$, we have that $(\mathscr{B} \setminus \mathscr{A}) + \mathscr{A} \in [\![I]\!]$, therefore $[\![I]\!] \models \mathscr{A}[\mathscr{B} \setminus \mathscr{A}]$, so that $[\![I]\!] \models \mathscr{A}[\mathscr{C}]$ for all C s.t. $\mathscr{A}' = \mathscr{B} \setminus \mathscr{A} \preccurlyeq \mathscr{C}$, because $[\![I]\!]$ is upward closed;

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

- if $I \Vdash G_1 \,\&\, G_2, \Delta[\mathscr{A}]$ then $\mathscr{A} = \mathscr{A}_1 \bullet \mathscr{A}_2$ and $I \Vdash G_1, \Delta[\mathscr{A}_1]$ and $I \Vdash G_2, \Delta[\mathscr{A}_2]$. By inductive hypothesis, $[\![I]\!] \models G_1, \Delta[\mathscr{B}_1]$ and $[\![I]\!] \models G_2, \Delta[\mathscr{B}_2]$ for any $\mathscr{B}_1, \mathscr{B}_2$ s.t. $\mathscr{A}_1 \preccurlyeq \mathscr{B}_1$ and $\mathscr{A}_2 \preccurlyeq \mathscr{B}_2$. That is, $[\![I]\!] \models G_i, \Delta[\mathscr{C}]$ for any $\mathscr{C} \in [\![\mathscr{A}_1 \bullet \mathscr{A}_2]\!]$ $i : 1, 2$. It follows that $[\![I]\!] \models G_1 \,\&\, G_2, \Delta[\mathscr{C}]$ for all $\mathscr{C} \in [\![\mathscr{A}_1 \bullet \mathscr{A}_2]\!]$;
- the $\wp$-case follows by a straightforward application of the inductive hypothesis.

(ii) By induction on Δ.

- The $\top$-case follows by definition;
- if $[\![I]\!] \models \mathscr{A}[\mathscr{A}']$ then $\mathscr{A}' + \mathscr{A} \in [\![I]\!]$, i.e. there exists $\mathscr{B} \in I$ s.t. $\mathscr{B} \preccurlyeq \mathscr{A}' + \mathscr{A}$. Since $\mathscr{B} \setminus \mathscr{A} \preccurlyeq (\mathscr{A}' + \mathscr{A}) \setminus \mathscr{A} = \mathscr{A}'$, it follows that for $\mathscr{C} = \mathscr{B} \setminus \mathscr{A}$, $I \Vdash \mathscr{A}[\mathscr{C}]$ and $\mathscr{C} \preccurlyeq \mathscr{A}'$;
- if $[\![I]\!] \models G_1 \,\&\, G_2, \Delta[\mathscr{A}]$ then $[\![I]\!] \models G_i, \Delta[\mathscr{A}]$ for $i : 1, 2$. By inductive hypothesis, there exists $\mathscr{A}_i$ such that $\mathscr{A}_i \preccurlyeq \mathscr{A}$, $I \Vdash G_i, \Delta[\mathscr{A}_i]$ for $i : 1, 2$, i.e., $I \Vdash G_1 \,\&\, G_2, \Delta[\mathscr{A}_1 \bullet \mathscr{A}_2]$. The thesis follows noting that $\mathscr{A}_1 \bullet \mathscr{A}_2 \preccurlyeq \mathscr{A}$;
- the $\wp$-case follows by a straightforward application of the inductive hypothesis.

(iii) If $I \Vdash \Delta[\mathscr{A}]$, then by *i*), $[\![I]\!] \models \Delta[\mathscr{A}]$. Since $[\![I]\!] \subseteq [\![J]\!]$ then, by Lemma 1 *ii*), $[\![J]\!] \models \Delta[\mathscr{A}]$. Thus, by *ii*), there exists $\mathscr{A}' \preccurlyeq \mathscr{A}$ s.t. $J \Vdash \Delta[\mathscr{A}']$.

(iv) By induction on Δ.

- If $\Delta = \top, \Delta'$, then, no matter which k you choose, $[\![I_k]\!] \models \top, \Delta'[\mathscr{A}]$;
- if Δ is a fact, then $\Delta, \mathscr{A} \in [\![\bigsqcup_{i=1}^{\infty} I_i]\!]$, that is there exists $\mathscr{B}$ s.t. $\mathscr{B} \in \bigsqcup_{i=1}^{\infty} I_i$ and $\mathscr{B} \preccurlyeq \Delta, \mathscr{A}$. Therefore there exists k s.t. $\mathscr{B} \in I_k$ and $\mathscr{B} \preccurlyeq \Delta, \mathscr{A}$, that is $\Delta, \mathscr{A} \in [\![I_k]\!]$;
- if $\Delta = G_1 \,\&\, G_2, \Delta'$, then by inductive hypothesis there exist k_1 and k_2 s.t. $[\![I_{k_1}]\!] \models G_1, \Delta'[\mathscr{A}]$ and $[\![I_{k_2}]\!] \models G_2, \Delta'[\mathscr{A}]$. Therefore, if $k = max\{k_1, k_2\}$, by Lemma 1 *ii*), we have that $[\![I_k]\!] \models G_1, \Delta'[\mathscr{A}]$ and $[\![I_k]\!] \models G_2, \Delta'[\mathscr{A}]$, therefore $[\![I_k]\!] \models G_1 \,\&\, G_2, \Delta'[\mathscr{A}]$, i.e. $[\![I_k]\!] \models \Delta[\mathscr{A}]$;
- the $\wp$-case follows by a straightforward application of the inductive hypothesis.

□

We are ready now to define the abstract fixpoint operator $S_P : \mathscr{I} \to \mathscr{I}$. We will proceed in two steps. We will first define an operator working over elements of $\mathscr{P}(B_P)$. With a little bit of overloading, we will call the operator with the same name, i.e., S_P. As for the S_P operator used in the symbolic semantics of CLP programs (Jaffar & Maher, 1994), the operator should satisfy the equation $[\![S_P(I)]\!] = T_P([\![I]\!])$ for any I, $J \in \mathscr{P}(B_P)$. This property ensures the soundness and completeness of the *symbolic* representation w.r.t. the ground semantics of LO programs.

After defining the operator over $\mathscr{P}(B_P)$, we will lift it to our abstract domain $\mathscr{I}$ consisting of the equivalence classes of elements of $\mathscr{P}(B_P)$ w.r.t. the relation $\simeq$ defined in Definition 5.1. Formally, we first introduce the following definition.

Definition 5.3 (*Symbolic Fixpoint Operator*)
Given an LO program P, and $I \in \mathscr{P}(B_P)$, the operator S_P is defined as follows:

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

The following property shows that S_P is sound and complete w.r.t. T_P.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Proposition 4
Let $I \in \mathscr{P}(B_P)$, then $[\![S_P(I)]\!] = T_P([\![I]\!])$.

Proof
Let $\mathscr{A} = \widehat{H}, \mathscr{B} \in S_P(I)$ where $H \circ\!\!- G \in P$ and $I \Vdash G[\mathscr{B}]$ then, by Lemma 2 *i*), $[\![I]\!] \models G[\mathscr{B}']$ for any $\mathscr{B}'$ s.t. $\mathscr{B} \preccurlyeq \mathscr{B}'$. Thus, for any $\mathscr{A}' = \widehat{H}, \mathscr{B}'$ s.t. $\mathscr{A} \preccurlyeq \mathscr{A}'$, $\mathscr{A}' \in T_P([\![I]\!])$.
Vice versa, if $\mathscr{A} \in T_P([\![I]\!])$ then $\mathscr{A} = \widehat{H}, \mathscr{B}$ where $H \circ\!\!- G \in P$ and $[\![I]\!] \models G[\mathscr{B}]$. By Lemma 2 *ii*), there exists $\mathscr{B}'$ s.t. $\mathscr{B}' \preccurlyeq \mathscr{B}$ and $I \Vdash G[\mathscr{B}']$, i.e. $\mathscr{A}' = \widehat{H}, \mathscr{B}' \in S_P(I)$ and $\mathscr{A}' \preccurlyeq \mathscr{A}$. □

Furthermore, the following corollary holds.

Corollary 2
Given $I, J \in \mathscr{P}(B_P)$, if $I \simeq J$ then $S_P(I) \simeq S_P(J)$.

Proof
If $I \simeq J$, then, by definition of $\simeq$, it follows that $[\![I]\!] = [\![J]\!]$. This implies that $T_P([\![I]\!]) = T_P([\![J]\!])$. Thus, by Prop. 4 it follows that $[\![S_P(I)]\!] = [\![S_P(J)]\!]$, i.e. $S_P(I) \simeq S_P(J)$. □

The previous corollary allows us to safely lift the definition of S_P from the lattice $\langle \mathscr{P}(B_P), \subseteq \rangle$ to the lattice $\langle \mathscr{I}, \sqsubseteq \rangle$. Formally, we define the abstract fixpoint operator as follows.

Definition 5.4 (*Abstract Fixpoint Operator* S_P)
Given an LO program P, and an equivalence class $[I]_\simeq$ of $\mathscr{I}$, the operator S_P is defined as follows:

$$S_P([I]_\simeq) = [S_P(I)]_\simeq$$

where $S_P(I)$ is defined in Definition 5.3.

In the following we will use I to denote its class $[I]_\simeq$. The abstract operator S_P satisfies the following property.

Proposition 5
S_P is monotonic and continuous over the lattice $\langle \mathscr{I}, \sqsubseteq \rangle$.

Proof
Monotonicity. For any $\mathscr{A} = \widehat{H}, \mathscr{B} \in S_P(I)$ there exists $H \circ\!\!- G \in P$ s.t. $I \Vdash G[\mathscr{B}]$. Assume now that $I \sqsubseteq J$. Then, by Lemma 2 *iii*), we have that $J \Vdash G[\mathscr{B}']$ for $\mathscr{B}' \preccurlyeq \mathscr{B}$. Thus, there exists $\mathscr{A}' = \widehat{H}, \mathscr{B}' \in S_P(J)$ such that $\mathscr{A}' \preccurlyeq \mathscr{A}$, i.e., $S_P(I) \sqsubseteq S_P(J)$.
Continuity. We show that S_P is finitary. Let $I_1 \sqsubseteq I_2 \sqsubseteq \ldots$ be an increasing sequence of interpretations. For any $\mathscr{A} = \widehat{H}, \mathscr{B} \in S_P(\bigsqcup_{i=1}^{\infty} I_i)$ there exists $H \circ\!\!- G \in P$ s.t. $\bigsqcup_{i=1}^{\infty} I_i \Vdash G[\mathscr{B}]$. By Lemma 2 *i*), $[\![\bigsqcup_{i=1}^{\infty} I_i]\!] \models G[\mathscr{B}]$. By Lemma 2 *iv*), we get that $[\![I_k]\!] \models G[\mathscr{B}]$ for some k, and by Lemma 2 *ii*), $I_k \Vdash G[\mathscr{B}']$ for $\mathscr{B}' \preccurlyeq \mathscr{B}$. Thus, $\mathscr{A}' = \widehat{H}, \mathscr{B}' \in S_P(I_k)$, i.e. $\mathscr{A}' \in \bigsqcup_{i=1}^{\infty} S_P(I_i)$, i.e. $S_P(\bigsqcup_{i=1}^{\infty} I_i) \sqsubseteq \bigsqcup_{i=1}^{\infty} S_P(I_i)$. □

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

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

Fig. 3. Symbolic fixpoint computation.

Corollary 3
$[\![lfp(S_P)]\!] = lfp(T_P)$.

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

Theorem 2 (*Soundness and Completeness*)
Given an LO program P, $O(P) = F(P) = [\![SymbF(P)]\!]$. Furthermore, there exists $k \in \mathrm{N}$ such that $SymbF(P) = \bigsqcup_{i=0}^{k} S_P{\uparrow_k}(\emptyset)$.

Proof
Theorem 1 and Corollary 3 show that $O(P) = F(P) = [\![SymbF(P)]\!]$. Corollary 1 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 used in Abdulla *et al.* (1996) and Finkel & Schnoebelen (2001) 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 Figure 3. Corollary 1 guarantees that the algorithm always terminates and returns a *symbolic* representation of $O(P)$. As a corollary of Theorem 2, we obtain the following result.

Corollary 4
The provability of $P \Rightarrow G$ in propositional LO is *decidable*.

In view of Proposition 1, this result can be considered as an instance of the general decidability result (Kopylov, 1995) for propositional *affine* linear logic (i.e. linear logic with *weakening*).

Example 3
We calculate the fixpoint semantics for the program P of Example 1, which is given below.

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

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

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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

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, as seen in Example 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 $S_P\uparrow_1 \Vdash \{b,c\}[\{d\}]$ and $S_P\uparrow_1 \Vdash \{b,c\}[\{f\}]$, therefore $\{d,e,e\}$ and $\{e,e,f\}$ belong to $S_P\uparrow_2$. We can therefore take the following equivalence class as representative for $S_P\uparrow_2$:

$$S_P\uparrow_2 = \{\{c,d\},\{c,f\},\{a,d\},\{a,f\},\{b,c\},\{d,e,e\},\{e,e,f\}\}.$$

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 $\{a,d\}$ and $\{a,f\}$ in $S_P\uparrow_2$ redundant. From clause 4 we can get that $\{e,e\}$ is another element of $S_P\uparrow_3$, which implies that the information given by $\{d,e,e\}$ and $\{e,e,f\}$ is now redundant. No additional (not redundant) elements are obtained from clause 2. We therefore can take

$$S_P\uparrow_3 = \{\{c,d\},\{c,f\},\{b,c\},\{a\},\{e,e\}\}.$$

The reader can verify that $S_P\uparrow_4 = S_P\uparrow_3 = SymbF(P)$ so that

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

We suggest the reader to compare the top-down proof for the goal e,e, given in Figure 2, and the part of the bottom-up computation which yields the same goal. The order in which the backchaining steps are performed is reversed, as expected. Moreover, the top-down computation requires to solve one goal, namely d,e,c, which is not *minimal*, in the sense that its proper subset c,d is still provable. Using the bottom-up algorithm sketched above, at every step only the minimal information (in this case c,d) is kept at every step. In general, this strategy has the further advantage of reducing the amount of non-determinism in the proof search. For instance, let us consider the goal b,e,e (which is certainly provable because of Proposition 1). This goal has at least two different proofs. The first is a slight modification of the proof in Figure 2 (just add the atom b to every sequent). An alternative proof is the following, obtained by changing the order of applications of the backchaining steps:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \Rightarrow e,b,\top}\,\top_r}{P \Rightarrow d,e,b,c}\,bc^{(3)}}{P \Rightarrow d,e,b \parr c}\,\parr_r}{P \Rightarrow d,e,e}\,bc^{(4)}}{P \Rightarrow d \parr e,e,e}\,\parr_r \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{P \Rightarrow b,\top}\,\top_r}{P \Rightarrow f,b,c}\,bc^{(5)}}{P \Rightarrow f,b \parr c}\,\parr_r}{P \Rightarrow f,e,e}\,bc^{(4)}}{P \Rightarrow (d \parr e)\,\&\,f,e,e}\,\&_r}{P \Rightarrow b,e,e}\,bc^{(2)}$$

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

There are even more complicated proofs (for instance in the left branch I could rewrite b again by backchaining over clause 2 rather than axiom 3). The bottom-up computation avoids these complications by keeping only *minimal* information at every step. We would also like to stress that the bottom-up computation is always guaranteed to terminate, as stated in Theorem 2, while in general the top-down computation can diverge. □

6 A bottom-up semantics for $\mathbf{LO_1}$

As shown in Andreoli (1992), 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 **1**. In other words, we extend LO with clauses of the following form:

$$A_1 \parr \ldots \parr A_n \multimap \mathbf{1}.$$

We name this language $\mathrm{LO_1}$, and use the notation $P \Rightarrow_{\mathbf{1}} \Delta$ for $\mathrm{LO_1}$ sequents. The meaning of the new kind of clauses is given by the following inference scheme:

$$\frac{}{P \Rightarrow_{\mathbf{1}} \mathbf{1}}\ \mathbf{1}_r$$

$$\frac{}{P \Rightarrow_{\mathbf{1}} \widehat{H}}\ bc\ (H \multimap \mathbf{1} \in P)$$

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 **1** intuitively introduces the possibility of *counting* resources.

Example 4

LO programs can be used to encode Petri Nets (see also the proof of Proposition 6 and Section 11). Let us consider a simple Petri net with three places a, b and c. We can represent a marking with a multiset of atoms and a transition with a clause. For instance, the clause $a \parr b \multimap c \parr c$ can be interpreted as the Petri Net transition that removes one token from place a, one token from place b, and adds two tokens to place c. By using the constant **1**, we can specify an operation *trans* which transfers *all* tokens in place a to place b. The encoding is as follows:

1. $a \parr trans \multimap b \parr trans$
2. $trans \multimap done \,\&\, check$
3. $check \parr b \multimap check$
4. $check \parr c \multimap check$
5. $check \multimap \mathbf{1}$

The first clause specifies the transfer of a single token from a to b, and it is supposed to be used as many times as the number of initial tokens in a. The second clause starts an auxiliary branch of the computation which checks that all tokens have been moved to b. The proof for the initial marking a, a, c is given in Figure 4 (where, for simplicity, applications of the $\parr_r$ and $\&_r$ rules have been incorporated into

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \Rightarrow_{\mathbf{1}} \mathbf{1}}\,\mathbf{1}_r}{P \Rightarrow_{\mathbf{1}} check}\,bc^{(5)}}{P \Rightarrow_{\mathbf{1}} c, check}\,bc^{(4)}}{P \Rightarrow_{\mathbf{1}} b, c, check}\,bc^{(3)}}{P \Rightarrow_{\mathbf{1}} b, b, c, check}\,bc^{(3)} \qquad \vdots \quad P \Rightarrow_{\mathbf{1}} b, b, c, done}{\dfrac{\dfrac{P \Rightarrow_{\mathbf{1}} b, b, c, trans}{P \Rightarrow_{\mathbf{1}} b, a, c, trans}\,bc^{(1)}}{P \Rightarrow_{\mathbf{1}} a, a, c, trans}\,bc^{(1)}}\,bc^{(2)}$$

Fig. 4. An $\mathrm{LO}_{\mathbf{1}}$ proof for the goal $a, a, c, trans$ in the program of Example 4.

backchaining steps). Note that the check cannot succeed if there are any tokens left in a. Using $\mathbf{1}$ in clause 5 is crucial to achieve this goal. □

Provability in $\mathrm{LO}_{\mathbf{1}}$ 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 $\mathrm{LO}_{\mathbf{1}}$ program as follows:

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

We first note that, in contrast with Proposition 1, the weakening rule is not admissible in $\mathrm{LO}_{\mathbf{1}}$. This implies that we cannot use the same techniques we used for the fragment without $\mathbf{1}$. So the question is: can we still find a finite representation of $O_{\mathbf{1}}(P)$? The following proposition gives us a negative answer.

Proposition 6
Given an $\mathrm{LO}_{\mathbf{1}}$ 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) as $\mathrm{LO}_{\mathbf{1}}$ programs. A VAS consists of 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 $x_i \geqslant -\delta_i$ ensure that the variables assume only positive values. Following (Cervesato, 1995), the encoding of a VAS in $\mathrm{LO}_{\mathbf{1}}$ 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 $\delta_i \geqslant 0$ (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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

post w.r.t. V and the initial markings). From classical results on Petri Nets (e.g. see the survey by Esparza & Nielsen (1994)), 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. □

Despite Proposition 6, it is still possible to define a *symbolic*, effective fixpoint operator for $\mathrm{LO}_\mathbf{1}$ programs as we will show in the following section. Before going into more details, we first rephrase the semantics of Section 4 for $\mathrm{LO}_\mathbf{1}$. We omit the proofs, which are analogous to those of Section 4. For the sake of simplicity, in the rest of the paper we will still denote the satisfiability judgments for $\mathrm{LO}_\mathbf{1}$ with $\models$ and $\Vdash$.

Definition 6.1 (*Satisfiability in* $LO_\mathbf{1}$)
Let I be a Herbrand interpretation, then $\models$ is defined as follows:

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

$$I \models \top, \Delta[\mathscr{A}'] \text{ for any fact } \mathscr{A}';$$

$$I \models \mathscr{A}[\mathscr{A}'] \text{ if } \mathscr{A} + \mathscr{A}' \in I;$$

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

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

The new satisfiability relation satisfies the following properties.

Lemma 3
For any interpretations I, J, context Δ, and fact $\mathscr{A}$,

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

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

Definition 6.2 (*Fixpoint operator* $T_P^\mathbf{1}$)
Given an $\mathrm{LO}_\mathbf{1}$ program P, and an interpretation I, the operator $T_P^\mathbf{1}$ is defined as follows:

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

The following property holds.

Proposition 7
$T_P^\mathbf{1}$ is monotonic and continuous over the lattice $\langle \mathscr{D}, \subseteq \rangle$.

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

Theorem 3 (*Soundness and Completeness*)
For every $\mathrm{LO}_\mathbf{1}$ program P, $F_\mathbf{1}(P) = O_\mathbf{1}(P)$.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

7 Constraint semantics for $\mathbf{LO_1}$

In this section we will define a *symbolic* fixpoint operator which relies on a constraint-based representation of provable multisets. The application of this operator is effective. Proposition 6 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 Proposition 6, 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 over integer variables. Namely, given a fact $\mathscr{A}$ we can denote its closure, i.e. the ideal $[\![\mathscr{A}]\!]$, by the constraint

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

Then all the operations on multisets involved in the definition of S_P (see Definition 5.2) can be expressed as operations over linear constraints. In particular, given the ideals $[\![\mathscr{A}]\!]$ and $[\![\mathscr{B}]\!]$, the ideal $[\![\mathscr{A} \bullet \mathscr{B}]\!]$ is represented as the constraint

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

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

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

where

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

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

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

where

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

The introduction of the constant $\mathbf{1}$ breaks down Proposition 1. As a consequence, the abstraction based on ideals is no more precise. In order to give a semantics for $\mathrm{LO_1}$, we need to add a class of constraints for representing collections of multisets which are not upward-closed (i.e. which are not ideals). We note then that we can represent a multiset $\mathscr{A}$ as the linear constraint

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

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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

bound of multisets) had in Definition 5.2, while $\exists \mathbf{x}'.(\varphi[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathscr{A}}(\mathbf{x}, \mathbf{x}'))$, for a given multiset $\mathscr{A}$, plays the role of multiset difference. The reader can compare Definition 5.2 with Definition 7.2. Based on these ideas, we can define a bottom-up evaluation procedure for $\mathrm{LO}_{\mathbf{1}}$ 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. $[\![\varphi]\!]_{\mathbf{1}} = \{\widehat{\mathbf{c}} \mid \mathbf{x} = \mathbf{c} \text{ satisfies } \varphi\}$.

Following Gabbrielli *et al.* (1995), 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 integer variables $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ associated to the signature $\Sigma = \{a_1, \ldots, a_n\}$. The operator $S_P^{\mathbf{1}}$ is defined on *constraint interpretations* consisting of sets (disjunctions) of (equivalence classes of) linear constraints. For brevity, we will define the semantics directly on the interpretations consisting of the representative elements of the equivalence classes. 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 $\langle \mathscr{C}, \subseteq \rangle$ with respect to set inclusion, where $\mathscr{C} = \mathscr{P}(LC_\Sigma)$.

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. $x = 0 \wedge x \geqslant 1$ has no solutions thus the following rules cannot be applied to this case).

Definition 7.2 (*Satisfiability in* $LO_{\mathbf{1}}$)
Let $I \in \mathscr{C}$, then $\Vdash$ is defined as follows:

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

$$I \Vdash \top, \Delta[\varphi] \text{ where } \varphi \equiv x_1 \geqslant 0 \wedge \ldots \wedge x_n \geqslant 0;$$

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

$$I \Vdash G_1 \parr G_2, \Delta[\varphi] \text{ if } I \Vdash G_1, G_2, \Delta[\varphi];$$

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

The relation $\Vdash$ satisfies the following properties.

Lemma 4
Given $I, J \in \mathscr{C}$,

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

(i) if $I \Vdash \Delta[\varphi]$, then $[\![I]\!]_{\mathbf{1}} \models \Delta[\mathscr{A}]$ for every $\mathscr{A} \in [\![\varphi]\!]_{\mathbf{1}}$;
(ii) if $[\![I]\!]_{\mathbf{1}} \models \Delta[\mathscr{A}]$, then there exists φ such that $I \Vdash \Delta[\varphi]$ and $\mathscr{A} \in [\![\varphi]\!]_{\mathbf{1}}$;
(iii) if $I \subseteq J$ and $I \Vdash \Delta[\varphi]$, then $J \Vdash \Delta[\varphi]$;
iv) 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. $I_k \Vdash \Delta[\varphi]$.

Proof

(i) By induction on Δ.

- If $I \Vdash_{\mathbf{1}} \top, \Delta[\varphi]$, then every $\mathbf{c}$ (with $c_i \geqslant 0$) is solution of φ, and $[\![I]\!]_{\mathbf{1}} \models \top, \Delta[\mathscr{A}']$ for every fact $\mathscr{A}'$;
- if $I \Vdash_{\mathbf{1}} \mathbf{1}[\varphi]$, then $\langle 0, \ldots, 0 \rangle$ is the only solution of φ, and $[\![I]\!]_{\mathbf{1}} \models \mathbf{1}[\epsilon]$;
- if $I \Vdash_{\mathbf{1}} \mathscr{A}[\varphi]$ then there exists $\psi \in I$ s.t. $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathscr{A}}(\mathbf{x}, \mathbf{x}'))$ is satisfiable. Then for every solution $\mathbf{c}$ of φ there exists a vector $\mathbf{c}'$ s.t. $\psi[\mathbf{c}'/\mathbf{x}]$ is satisfiable and $c'_1 \geqslant Occ_{\mathscr{A}}(a_1), c_1 = c'_1 - Occ_{\mathscr{A}}(a_1), \ldots, c'_n \geqslant Occ_{\mathscr{A}}(a_n), c_n = c'_n - Occ_{\mathscr{A}}(a_n)$. From this we get that for $i = 1, \ldots, n$, $c'_i = c_i + Occ_{\mathscr{A}}(a_i)$ is a solution for ψ, therefore $\widehat{\mathbf{c}} + \mathscr{A} \in [\![\psi]\!]_{\mathbf{1}} \subseteq [\![I]\!]_{\mathbf{1}}$ so that we can conclude $[\![I]\!]_{\mathbf{1}} \models \mathscr{A}[\widehat{\mathbf{c}}]$;
- if $I \Vdash_{\mathbf{1}} G_1 \,\&\, G_2, \Delta[\varphi]$ then $\varphi \equiv \varphi_1 \wedge \varphi_2$ and $I \Vdash_{\mathbf{1}} G_1, \Delta[\varphi_1]$, $I \Vdash_{\mathbf{1}} G_2, \Delta[\varphi_2]$. By inductive hypothesis, $[\![I]\!]_{\mathbf{1}} \models G_1, \Delta[\widehat{\mathbf{c}_1}]$ and $[\![I]\!]_{\mathbf{1}} \models G_2, \Delta[\widehat{\mathbf{c}_2}]$ for every $\mathbf{c_1}$ and $\mathbf{c_2}$ solutions of φ_1 and φ_2, respectively. Thus $[\![I]\!]_{\mathbf{1}} \models G_1 \,\&\, G_2, \Delta[\widehat{\mathbf{c}}]$ for every $\mathbf{c}$ which is solution of both φ_1 and φ_2, i.e. for every $\mathbf{c}$ which is solution of $\varphi_1 \wedge \varphi_2$;
- the $\otimes$-case follows by a straightforward application of the inductive hypothesis.

(ii) By induction on Δ.

- $[\![I]\!]_{\mathbf{1}} \models \top, \Delta[\widehat{\mathbf{c}}]$ for every $\mathbf{c}$, and $I \Vdash_{\mathbf{1}} \top, \Delta[\varphi]$, where $\varphi \equiv x_1 \geqslant 0, \ldots, x_n \geqslant 0$, and every $\mathbf{c}$ is solution of φ;
- if $[\![I]\!]_{\mathbf{1}} \models \mathbf{1}[\epsilon]$, $\epsilon = \overbrace{\langle 0, \ldots, 0 \rangle}$, then $I \Vdash_{\mathbf{1}} \mathbf{1}[\varphi]$, where $\varphi \equiv x_1 = 0, \ldots, x_n = 0$, and $\langle 0, \ldots, 0 \rangle$ is solution of φ;
- if $[\![I]\!]_{\mathbf{1}} \models \mathscr{A}[\widehat{\mathbf{c}}]$ then $\widehat{\mathbf{c}} + \mathscr{A} \in [\![I]\!]_{\mathbf{1}}$, therefore there exists $\psi \in I$ s.t. $\widehat{\mathbf{c}} + \mathscr{A} \in [\![\psi]\!]_{\mathbf{1}}$. Therefore, if $\mathbf{a}$ is such that $\widehat{\mathbf{a}} = \mathscr{A}$, we have that $\psi[\mathbf{c} + \mathbf{a}/x]$ is satisfiable, $\mathbf{c}$ is solution of $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathscr{A}}(\mathbf{x}, \mathbf{x}'))$ and $I \Vdash_{\mathbf{1}} \mathscr{A}[\varphi]$;
- if $[\![I]\!]_{\mathbf{1}} \models G_1 \,\&\, G_2, \Delta[\widehat{\mathbf{c}}]$ then $[\![I]\!]_{\mathbf{1}} \models G_1, \Delta[\widehat{\mathbf{c}}]$ and $[\![I]\!]_{\mathbf{1}} \models G_2, \Delta[\widehat{\mathbf{c}}]$. By inductive hypothesis, there exist φ_1 and φ_2 such that $I \Vdash_{\mathbf{1}} G_1, \Delta[\varphi_1]$ and $I \Vdash_{\mathbf{1}} G_2, \Delta[\varphi_2]$, and $\mathbf{c}$ is a solution of φ_1 and φ_2. Therefore $I \Vdash_{\mathbf{1}} G_1 \,\&\, G_2, \Delta[\varphi_1 \wedge \varphi_2]$ and $\mathbf{c}$ is a solution of $\varphi_1 \wedge \varphi_2$;
- the $\otimes$-case follows by a straightforward application of the inductive hypothesis.

(iii) By simple induction on Δ.

(iv) By induction on Δ.

- The $\top$ and $\mathbf{1}$-cases follow by definition;
- if $\bigcup_{i=1}^{\infty} I_i \Vdash_{\mathbf{1}} \mathscr{A}[\varphi]$, then there exists $\psi \in \bigcup_{i=1}^{\infty} I_i$ s.t. $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \rho_{\mathscr{A}}(\mathbf{x}, \mathbf{x}'))$ is satisfiable. Then there exists k s.t. $\psi \in I_k$ and $I_k \Vdash_{\mathbf{1}} \mathscr{A}[\varphi]$;
- if $\bigcup_{i=1}^{\infty} I_i \Vdash_{\mathbf{1}} G_1 \,\&\, G_2, \Delta[\varphi]$, then $\varphi \equiv \varphi_1 \wedge \varphi_2$, and, by inductive hypothesis, there exist k_1 and k_2 s.t. $I_{k_1} \Vdash_{\mathbf{1}} G_1, \Delta[\varphi_1]$ and $I_{k_2} \Vdash_{\mathbf{1}} G_2, \Delta[\varphi_2]$. Then, for $k = max\{k_1, k_2\}$, we have, by *iii*), $I_k \Vdash_{\mathbf{1}} G_1, \Delta[\varphi_1]$ and $I_k \Vdash_{\mathbf{1}} G_2, \Delta[\varphi_2]$, therefore $I_k \Vdash_{\mathbf{1}} G_1 \,\&\, G_2, \Delta[\varphi_1 \wedge \varphi_2]$;

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

– the $\otimes$-case follows by a straightforward application of the inductive hypothesis.

□

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

Definition 7.3 (*Symbolic Fixpoint Operator* $S_P^{\mathbf{1}}$)
Given an $\mathrm{LO}_{\mathbf{1}}$ program P, and $I \in \mathscr{C}$, the operator $S_P^{\mathbf{1}}$ is defined as follows:

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

The new operator satisfies the following property.

Proposition 8
The operator $S_P^{\mathbf{1}}$ is monotonic and continuous over the lattice $\langle \mathscr{C}, \subseteq \rangle$.

Proof
Monotonicity. Immediate from $S_P^{\mathbf{1}}$ definition and Lemma 4 (iii).
Continuity. Let $I_1 \subseteq I_2 \subseteq \ldots$, be an increasing sequence of interpretations. We show that $S_P^{\mathbf{1}}(\bigcup_{i=1}^{\infty} I_i) \subseteq \bigcup_{i=1}^{\infty} S_P^{\mathbf{1}}(I_i)$. If $\varphi \in S_P^{\mathbf{1}}(\bigcup_{i=1}^{\infty} I_i)$, by definition there exists a clause $H \circ\!\!- G \in P$ s.t. $\bigcup_{i=1}^{\infty} I_i \Vdash_{\mathbf{1}} G[\psi]$ and $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \alpha_{\widehat{H}}(\mathbf{x}, \mathbf{x}'))$ is satisfiable. By Lemma 4 *iv*), there exists k s.t. $I_k \Vdash_{\mathbf{1}} G[\psi]$. This implies that $\varphi \in S_P^{\mathbf{1}}(I_k)$, i.e. $\varphi \in \bigcup_{i=1}^{\infty} S_P^{\mathbf{1}}(I_i)$. □

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

Proposition 9
Let $I \in \mathscr{C}$, then $[\![S_P^{\mathbf{1}}(I)]\!]_{\mathbf{1}} = T_P^{\mathbf{1}}([\![I]\!]_{\mathbf{1}})$.

Proof
Let $\widehat{\mathbf{c}} \in [\![S_P^{\mathbf{1}}(I)]\!]_{\mathbf{1}}$, then there exist $\varphi \in S_P^{\mathbf{1}}(I)$ and a clause $H \circ\!\!- G \in P$, s.t. $\mathbf{c}$ is solution of φ, $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \alpha_{\widehat{\mathscr{H}}}(\mathbf{x}, \mathbf{x}'))$ and $I \Vdash_{\mathbf{1}} G[\psi]$. Then there exists $\mathbf{c}'$ solution of ψ s.t. $\widehat{\mathbf{c}} = \widehat{\mathbf{c}'} + \widehat{H}$, and, by Lemma 4 *i*), $[\![I]\!]_{\mathbf{1}} \models G[\widehat{\mathbf{c}'}]$. Therefore, by definition of $T_P^{\mathbf{1}}$, $\widehat{\mathbf{c}} = \widehat{\mathbf{c}'} + \widehat{H} \in T_P^{\mathbf{1}}([\![I]\!]_{\mathbf{1}})$. Vice versa, let $\widehat{\mathbf{c}} \in T_P^{\mathbf{1}}([\![I]\!]_{\mathbf{1}})$, then there exists $H \circ\!\!- G \in P$ s.t. $[\![I]\!]_{\mathbf{1}} \models G[\mathscr{A}]$ and $\widehat{\mathbf{c}} = \widehat{H} + \mathscr{A}$. By Lemma 4 *ii*), there exists ψ s.t. $I \Vdash_{\mathbf{1}} G[\psi]$ and $\mathscr{A} \in [\![\psi]\!]_{\mathbf{1}}$. Therefore $\varphi \equiv \exists \mathbf{x}'.(\psi[\mathbf{x}'/\mathbf{x}] \wedge \alpha_{\widehat{H}}(\mathbf{x}, \mathbf{x}')) \in S_P^{\mathbf{1}}(I)$, and $\widehat{\mathbf{c}} = \widehat{H} + \mathscr{A} \in [\![\varphi]\!]_{\mathbf{1}} \subseteq [\![S_P^{\mathbf{1}}(I)]\!]_{\mathbf{1}}$. □

Corollary 5
$[\![lfp(S_P^{\mathbf{1}})]\!]_{\mathbf{1}} = lfp(T_P^{\mathbf{1}})$.

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

Theorem 4 (*Soundness and completeness*)
Given an $\mathrm{LO}_{\mathbf{1}}$ program P, $O_{\mathbf{1}}(P) = F_{\mathbf{1}}(P) = [\![SymbF_{\mathbf{1}}(P)]\!]_{\mathbf{1}}$.

Proof
By Theorem 3 and Corollary 5. □

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

8 Bottom-up evaluation for $\mathbf{LO_1}$

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 Proposition 6, the termination of the algorithm is not guaranteed a priori. In this respect, Theorem 2 gives us sufficient conditions that ensure its termination. The symbolic fixpoint operator $S_P^\mathbf{1}$ of Section 7 is defined over the lattice $\langle \mathscr{P}(LC_\Sigma), \subseteq \rangle$, with set inclusion being the partial order relation and set union the least upper bound operator. When we come to a concrete implementation of $S_P^\mathbf{1}$, 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 $\mathbf{x} \geqslant \mathbf{c}$, 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 5

We calculate the fixpoint semantics for the following $\mathrm{LO}_\mathbf{1}$ program made up of six clauses:

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

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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

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

Fig. 5. Symbolic fixpoint computation for the program in Example 5.

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 Figure 5).

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 $x_a \geqslant 0 \wedge x_b \geqslant 3 \wedge x_c \geqslant 0$. By 5, we have $x_a \geqslant 2 \wedge x_b \geqslant 0 \wedge x_c \geqslant 0$, 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 Figure 5.

Now, we can compute $S_P\uparrow_3$. By 4 and $x_a \geqslant 2 \wedge x_b \geqslant 0 \wedge x_c \geqslant 0 \in S_P\uparrow_2$ we get $x_a \geqslant 1 \wedge x_b \geqslant 2 \wedge x_c \geqslant 0$, which is subsumed by $x_a \geqslant 1 \wedge x_b \geqslant 1 \wedge x_c \geqslant 0$. 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 $x_a \geqslant 1 \wedge x_b \geqslant 1 \wedge x_c \geqslant 0$. Similarly, by 5. and $x_a \geqslant 0 \wedge x_b \geqslant 3 \wedge x_c \geqslant 0$ we get redundant information. By 6, from $x_a \geqslant 1 \wedge x_b \geqslant 1 \wedge x_c \geqslant 0$ 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 $x_a \geqslant 1 \wedge x_b \geqslant 1 \wedge x_c \geqslant 0$ and $x_a \geqslant 0 \wedge x_b \geqslant 3 \wedge x_c \geqslant 0$ we get $x_a \geqslant 0 \wedge x_b \geqslant 2 \wedge x_c \geqslant 1$, and finally from $x_a \geqslant 2 \wedge x_b \geqslant 0 \wedge x_c \geqslant 0$ and $x_a \geqslant 1 \wedge x_b \geqslant 1 \wedge x_c \geqslant 0$ we have $x_a \geqslant 1 \wedge x_b \geqslant 0 \wedge x_c \geqslant 1$. 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 $S_P\uparrow_4 = S_P\uparrow_3 = SymbF_{\mathbf{1}}(P)$, 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 $[\![\mathscr{A}]\!] = \{\mathscr{B} \mid \mathscr{A} \preccurlyeq \mathscr{B}\}$, where $\preccurlyeq$ is multiset inclusion):

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

□

It is often not the case that the symbolic computation of $\mathrm{LO}_{\mathbf{1}}$ 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 $\mathrm{LO}_{\mathbf{1}}$ programs as a least fixpoint computation over infinite-state *integer* systems allows us to apply techniques and tools developed in *infinite-state* model checking (Abdulla *et al.*, 1996; Bultan *et al.*, 1997; Delzanno & Podelski, 1999; Finkel & Schnoebelen, 2001; Henzinger *et al.*, 1997) and program analysis (Cousot & Halbwachs, 1978) to compute approximations of the least fixpoint of $S_P^{\mathbf{1}}$.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

In the next section we will present an interesting application of the semantical framework we have presented so far. Namely, we shall make a detailed comparison between LO and Disjunctive Logic Programming. This will help us in clarifying the relations and the relative strength of the languages. After recalling the basic definitions of DLP in Section 9, we will present our view of DLP as an abstraction of LO in Section 10. Finally, in Section 11 we will give a few hints on how to employ this framework to study reachability problems in Petri Nets.

9 An application of the semantics: Relation with DLP

As anticipated in the introduction, the paradigms of linear logic programming and Disjunctive Logic Programming have in common the use of *multi-headed* clauses. However, the operational interpretation of the extended notion of clause is quite different in the two paradigms. In fact, as shown in Bozzano *et al.* (2000b), from a proof-theoretical perspective it is possible to view LO as a *sub-structural* fragment of DLP in which the rule of *contraction* is forbidden on the right-hand side of sequents.

While proof theory allows one to compare the *top-down* semantics of the two languages, abstract interpretation (Cousot & Cousot, 1977) can be used to relate the fixpoint, bottom-up evaluation of programs. In the following we will focus on the latter approach, exploiting our semantics of LO and the bottom-up semantics of DLP given in Minker *et al.* (1991) . For the sake of clarity, we will use superscripts in order to distinguish between the fixpoint operators for LO and DLP, which will be denoted by T_P^{lo} and T_P^{dlp}, respectively. First of all, we recall some definitions concerning Disjunctive Logic Programming.

A *disjunctive logic program* as defined in Minker *et al.* (1991) is a finite set of clauses

$$A_1 \vee \ldots \vee A_n \leftarrow B_1 \wedge \ldots \wedge B_m,$$

where $n \geqslant 1$, $m \geqslant 0$, and A_i and B_i are atomic formulas. A *disjunctive goal* is of the form $\leftarrow C_1, \ldots, C_n$, where C_i is a *positive clause* (i.e. a disjunction of atomic formulas) for $i : 1, \ldots, n$. To make the language symmetric, in this paper we will consider extended clauses of the form

$$A_1 \vee \ldots \vee A_n \leftarrow C_1 \wedge \ldots \wedge C_m$$

containing positive clauses in the body. Following Minker *et al.* (1991) , we will identify positive clauses with sets of atoms. In order to define the operational and denotational semantics of DLP, we need the following notions.

Definition 9.1 (*Disjunctive Herbrand Base*)
The *disjunctive Herbrand base* of a program P, for short DHB_P, is the set of all positive clauses formed by an arbitrary number of atoms.

Definition 9.2 (*Disjunctive Interpretation*)
A subset I of the disjunctive Herbrand base DHB_P is called a disjunctive Herbrand interpretation.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Definition 9.3 (*Ground SLO-derivation*)
Let P be a DLP program. An SLO-derivation of a ground goal G from P consists of a sequence of goals $G_0 = G, G_1, \ldots$ such that for all $i \geqslant 0$, G_{i+1} is obtained from $G_i = \leftarrow (C_1, \ldots, C_m, \ldots, C_k)$ as follows:

- $C \leftarrow D_1 \wedge \ldots \wedge D_q$ is a ground instance of a clause in P such that C is contained in C_m (the selected clause);
- G_{i+1} is the goal $\leftarrow (C_1, \ldots, C_{m-1}, D_1 \vee C_m, \ldots, D_q \vee C_m, C_{m+1}, \ldots, C_k)$.

Definition 9.4 (*SLO-refutation*)
Let P be a DLP program. An SLO-refutation of a ground goal G from P is an SLO-derivation $G_0, G_1, \ldots, G_k$ s.t. G_k consists of the empty clause only.

As SLD-resolution for Horn programs, SLO-resolution gives us a procedural interpretation of DLP programs. The operational semantics is defined then as follows:

$$O_P^{dlp} = \{C \mid C \in DHB_P,\ \leftarrow C \text{ has an SLO-refutation}\}.$$

As for Horn programs, it is possible to define a fixpoint semantics via the following operator (where $gnd(P)$ denotes the set of ground instances of clauses in P).

Definition 9.5 (*The T_P^{dlp} Operator*)
Given a DLP program P and $I \subseteq DHB_P$,

$$\begin{aligned} T_P^{dlp}(I) \quad = \quad & \{\ C \in DHB_P \mid C' \leftarrow D_1, \ldots, D_n \in gnd(P), \\ & D_i \vee C_i \in I,\ i : 1, \ldots, n \\ & C = C' \vee C_1 \vee \ldots \vee C_n\ \}. \end{aligned}$$

The operator T_P^{dlp} is monotonic and continuous on the lattice of interpretations ordered w.r.t. set inclusion. Based on this property, the fixpoint semantics is defined as $F_P^{dlp} = lfp(T_P^{dlp}) = T_P^{dlp}\uparrow_\omega$. As shown in Minker *et al.* (1991) , for all $C \in O_P^{dlp}$ there exists $C' \in F_P^{dlp}$ s.t. C' implies C. Note that for two ground clauses C and C', C implies C' if and only if $C \subseteq C'$. This suggests that interpretations can also be ordered w.r.t. subset inclusion for their elements, i.e. $I \sqsubseteq J$ if and only if for all $A \in I$ there exists $B \in J$ such that $B \subseteq A$ (B implies A). In the rest of the paper we will consider this latter ordering.

Example 6
Consider the disjunctive program $P = \{r(a),\ p(X) \vee q(X) \leftarrow r(X)\}$ and the auxiliary predicate t. Then, $DHB_P = \{r(a), p(a), q(a), t(a), p(a) \vee r(a), p(a) \vee q(a), p(a) \vee q(a) \vee r(a), \ldots\}$. Furthermore, the goal $G_0 = \leftarrow (p(a) \vee q(a) \vee t(a))$ has the refutation $G_0, G_1 = \leftarrow (p(a) \vee q(a) \vee t(a) \vee r(a)), G_2$ where G_2 consists of the empty clause only. The fixpoint semantics of P is as follows $F_P^{dlp} = \{r(a),\ p(a) \vee q(a)\}$. Note that $p(a) \vee q(a) \vee t(a)$ is implied by $p(a) \vee q(a)$. □

We note that the definition of the T_P^{dlp} operator can be re-formulated in such a way that its input and output domains contain multisets instead of sets of atoms (i.e. we can consider interpretations which are sets of multisets of atoms). In fact, we can always map a multiset to its *underlying set*, i.e. the set containing the elements

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

with multiplicity greater than zero, and, vice versa, a set can be viewed as a multiset in which each element has multiplicity equal to one. In the following we will assume that T_P^{dlp} is defined on domains containing multisets. As the fixpoint operator for LO is defined on the same kind of domains, this will make the comparison between the two operators easier. Furthermore, without loss of generality, we will make the assumption that in clauses like $A_1 \vee \ldots \vee A_n \leftarrow C_1 \wedge \ldots \wedge C_m$, the A_i's are all *distinct* and each C_j consist of *distinct* atoms. This will simplify the embedding of DLP clauses into linear logic. The previous definitions can be easily adapted.

Now, we give a closer look at the formal presentations of DLP and LO. As said in the Introduction, we only need to consider a fragment of LO in which connectives can not be arbitrarily nested in goals, like in DLP. This fragment can be described by the following grammar:

$$\begin{aligned} \mathbf{H} &::= \mathbf{A}_1 \parr \ldots \parr \mathbf{A}_n \\ \mathbf{D} &::= \mathbf{H} \circ\!\!- \mathbf{G} \mid \mathbf{D} \,\&\, \mathbf{D} \\ \mathbf{G} &::= \mathbf{H}_1 \,\&\, \ldots \,\&\, \mathbf{H}_n \mid \top \end{aligned}$$

where $\mathbf{A}_i$ is an atomic formula. The comparison between the two languages is based on the idea that, to some extent, linear connectives, i.e. additive conjunction & and multiplicative disjunction $\parr$, should behave like classical conjunction $\wedge$ and classical disjunction $\vee$. Actually, classical connectives give rise to a fixpoint semantics for DLP which is a proper abstraction of the semantics for LO. The translation between linear and classical connectives is given via the following mapping $\lceil\cdot\rceil$:

$$\lceil F \vee G \rceil = \lceil F \rceil \parr \lceil G \rceil,\ \lceil F \wedge G \rceil = \lceil F \rceil \,\&\, \lceil G \rceil,\ \lceil F \leftarrow G \rceil = \lceil F \rceil \circ\!\!- \lceil G \rceil,\ \lceil \mathtt{tt} \rceil = \top$$

To make the comparison between DLP and LO more direct, it is possible to present DLP by means of the following grammar:

$$\begin{aligned} \mathbf{H} &::= \mathbf{A}_1 \vee \ldots \vee \mathbf{A}_n \\ \mathbf{D} &::= \mathbf{H} \leftarrow \mathbf{G} \mid \mathbf{D} \wedge \mathbf{D} \\ \mathbf{G} &::= \mathbf{H}_1 \wedge \ldots \wedge \mathbf{H}_n \mid \mathtt{tt} \end{aligned}$$

where $\mathbf{A}_i$ is an atomic formula. A DLP program P is now a **D**-clause, whereas DLP goals are represented (modulo '$\leftarrow$') as **G**-formulas. Here, we have introduced an explicit constant `tt` for *true* and we have written *unit clauses* (i.e. with empty body) with the syntax $A_1 \vee \ldots \vee A_n \leftarrow \mathtt{tt}$. With these conventions, the grammars for LO and DLP given above are exactly the same modulo the translation $\lceil\cdot\rceil$. The definitions concerning the operational and fixpoint semantics for DLP given previously can be adapted in a straightforward manner. The reader can also note that by definition of DLP program, the image of $\lceil\cdot\rceil$ returns a class of LO programs where both the head and the disjuncts in the body have no repeated occurrences of the same atom.

We conclude this section by specializing our fixpoint semantics for LO, given in Section 5, to the simpler fragment presented above. We give the following definition for the T_P^{lo} operator:

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Definition 9.6 (T_P^{lo} *operator*)
Given an LO program P and an interpretation I, the operator T_P^{lo} is defined as follows:

$$T_P^{lo}(I) = \{\widehat{H} + (\mathscr{C}_1 \bullet \ldots \bullet \mathscr{C}_n) \mid H \circ\!\!- D_1 \,\&\, \ldots \,\&\, D_n \in P,\ \forall i = 1, \ldots, n,\ \widehat{D}_i + \mathscr{C}_i \in I\} \cup \{\widehat{H} \mid H \circ\!\!- \top \in P\}$$

The operator T_P^{lo} is monotonic and continuous over the lattice of Herbrand interpretations (ordered w.r.t. $\sqsubseteq$). Thus, the fixpoint semantics of an LO-program P is defined as

$$F_P^{lo} = \bigsqcup_{i=0}^{\omega} T_P^{lo}\uparrow_i .$$

A completeness result similar to that of Section 5, stating the equivalence between the operational and fixpoint semantics, obviously holds for the fragment of LO considered here.

10 DLP as abstraction of LO

The fixpoint semantics of LO allows us to investigate in more depth the relationships between LO and DLP. For this purpose, we can employ the mathematical tools provided by *abstract interpretation* (Cousot & Cousot, 1977), and in particular the notion of *completeness* (Cousot & Cousot, 1977; Giacobazzi & Ranzato, 1997) that qualifies the precision of an abstraction. Informally, the comparison between LO and DLP fixpoint semantics is based on the abstraction that maps multisets into sets of atomic formulas (positive clauses). This abstraction induces a Galois connection between the semantic domains of DLP and LO. We prove that the fixpoint semantics of the translation of an LO program in DLP is a correct abstraction of the fixpoint semantics of the original LO program. Furthermore, we show that this abstraction is not *fully complete* with respect to LO semantics. In a *fully complete* abstraction the result of interleaving the application of the abstract fixpoint operator with the abstraction α coincides with the abstraction of the concrete fixpoint operator. For a *complete* abstraction, a similar relation holds for fixpoints, i.e. the fixpoint of the abstract operator coincides with the abstraction of the fixpoint of the concrete one. We isolate an interesting class of LO programs for which we show that the property of completeness holds. In particular, completeness holds if we forbid conjunctions in the body of clauses. The resulting class of LO programs is still very interesting, as it can be used to encode Petri Nets.

Abstract Interpretation (Cousot & Cousot, 1977, 1979) is a classical framework for semantics approximation which is used for the construction of semantics-based program analysis algorithms. Given a semantics and an abstraction of the language constructors and standard data, abstract interpretation determines an abstract representation of the language which is, by construction, sound with respect to the standard semantics. This new representation enables the calculation of the abstract semantics in finite time, although it implies some loss of precision. We recall here some key concepts in abstract interpretation, which the reader can find in Cousot & Cousot (1977, 1979) and Giacobazzi & Ranzato (1997).

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Given a concrete semantics $\langle C, T_P \rangle$, specified by a *concrete domain* (complete lattice) C and a (monotone) fixpoint operator $T_P : C \to C$, the abstract semantics can be specified by an *abstract domain* A and an abstract fixpoint operator $T_P^{\#}$. In this context, program semantics is given by $lfp(T_P)$, and its abstraction is $lfp(T_P^{\#})$. The concrete and abstract semantics $S = \langle C, T_P \rangle$ and $S^{\#} = \langle A, T_P^{\#} \rangle$ are related by a Galois connection $\langle \alpha, C, A, \gamma \rangle$, where $\alpha : C \to A$ and $\gamma : A \to C$ are called *abstraction* and *concretization* functions, respectively. $S^{\#}$ is called a *sound* abstraction of S if for all P, $\alpha(lfp(T_P)) \leqslant_A lfp(T_P^{\#})$. This condition is implied by the strongest property of *full soundness*, which requires that $\alpha \circ T_P \leqslant_A T_P^{\#} \circ \alpha$. The notions of *completeness* and *full completeness* are dual with respect to those of soundness. Namely, $S^{\#}$ is a (fully) *complete* abstraction of S if for all P, $(T_P^{\#} \circ \alpha \leqslant_A \alpha \circ T_P)$ $lfp(T_P^{\#}) \leqslant_A \alpha(lfp(T_P))$. Often, the notion of completeness is assumed to include soundness (i.e. we impose '=' in the previous equations). It is well-known (Cousot & Cousot, 1979) that the abstract domain A induces a *best* correct approximation of T_P, which is given by $\alpha \circ T_P \circ \gamma$, and that it is possible to define a (fully) complete abstract operator $T_P^{\#}$ if and only if the best correct approximation is (fully) complete. It can be proved that for a fixed concrete semantics, (full) completeness of an abstract interpretation only depends on the choice of the abstract domain. The problem of achieving a (fully) complete abstract interpretation starting from a correct one, by either refining or simplifying the abstract domain, is studied in Giacobazzi & Ranzato (1997). We conclude by observing that an equivalent presentation of abstract interpretation is based on *closure operators* (Cousot & Cousot, 1979), i.e. functions from a concrete domain C to itself which are *monotone*, *idempotent* and *extensive*. This approach provides independence from specific representations of abstract domain's objects (the abstract domain is given by the image, i.e. the set of fixpoints, of the closure operator).

We are now in the position of connecting the LO (concrete) semantics with the DLP (abstract) semantics. We define the abstract interpretation as a closure operator on the lattice $\mathscr{I}$, the domain of LO interpretations of Definition 5.1. In fact, as mentioned before, we can consider disjunctive interpretations as a subclass of $\mathscr{I}$ (i.e. all sets in $\mathscr{I}$). We recall that in $\mathscr{I}$ the ordering of interpretations is defined as follows: $I \sqsubseteq J$ iff for all $B \in I$ there exists $A \in J$ such that A is a sub-multiset of B (i.e. for disjunctive interpretations, $A \subseteq B$). We give the following definitions.

Definition 10.1 (*Abstract Interpretation from LO to DLP*)
The abstract interpretation is defined by the closure operator $\alpha : \mathscr{I} \to \mathscr{I}$ such that for every $I \in \mathscr{I}$, $\alpha(I) = \{\alpha(\mathscr{A}) \mid \mathscr{A} \in \mathscr{I}\}$, where for a given multiset $\mathscr{A}$, $\alpha(\mathscr{A})$ is the multiset such that for every $i = 1, \ldots, n$, $Occ_{\alpha(\mathscr{A})}(a_i) = 0$ if $Occ_{\mathscr{A}}(a_i) = 0$, $Occ_{\alpha(\mathscr{A})}(a_i) = 1$ otherwise (i.e., we abstract a multiset with the corresponding set).

Definition 10.2 (*Abstract semantics*)
The abstract fixpoint semantics is given by $lfp(T_P^{\#})$, where the abstract fixpoint operator $T_P^{\#}$ is defined as $(\alpha \circ T_P^{lo})$.

According to Cousot & Cousot (1979), $\alpha \circ T_P^{lo}$ is the best correct approximation of the concrete fixpoint operator T_P^{lo}, for the fixed abstraction α. The abstraction α,

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

as said before, transforms multisets into sets by forgetting multiple occurrences of atoms. It is not difficult to convince ourselves that $T_P^{\#}$ is indeed the T_P^{dlp} operator for disjunctive logic programs, provided that, as discussed in Section 9, we consider T_P^{dlp} defined over domains containing multisets instead of sets (actually, we are identifying T_P^{dlp} input domain with the abstract domain which is given by the set of fixpoints, i.e., the image, of the closure operator α). The operations $\bullet$ (*least upper bound* of multisets) and $+$ (multiset union) used in the definition of T_P^{lo} are interchangeable (because of the subsequent application of the operator α) and correspond to set (multiset) union in the definition of T_P^{dlp}. We have the following results.

Proposition 10 (*DLP is an abstraction of LO*)
For every DLP program P and *disjunctive* (hence LO) interpretation I, $T_P^{dlp}(I) = T^{\#}_{\lceil P \rceil}(I)$.

Proof
By definitions. □

Proposition 11 (*DLP is a correct abstraction of LO*)
For every LO program P, the abstract semantics is a fully sound abstraction of the concrete semantics, that is, for every interpretation I, $\alpha(T_P^{lo}(I)) \sqsubseteq T_P^{\#}(\alpha(I))$.

Proof
As $T_P^{\#} = \alpha \circ T_P^{lo}$ and $I \sqsubseteq \alpha(I)$, the proposition follows by monotonicity of $T_P^{\#}$. □

The previous result implies *soundness*, i.e. $\alpha(lfp(T_P^{lo})) \sqsubseteq lfp(T_P^{\#})$. The strong property of *full completeness* does not hold for the abstraction. To see why, take as a counterexample the single clause $a \circ\!\!- b$ and the interpretation I with the single multiset $\{b, b\}$. Then, $\alpha(T_P^{lo}(I)) = \{\{a, b\}\}$, $T_P^{\#}(\alpha(I)) = \{\{a\}\}$, and $T_P^{\#}(\alpha(I)) \not\sqsubseteq \alpha(T_P^{lo}(I))$.

We conclude this section showing that the abstraction is complete for the subclass of LO programs whose clauses contain *at most* one conjunct in the body. We will address some applications of this result in Section 11.

(Note: the abstraction not being fully complete has a counterpart in the fact that in general $h > 1$ in the following lemma, i.e. more than one step of T_P^{lo} is necessary to simulate one step of $T_P^{\#}$.)

Lemma 5
Let P be an LO program in which every clause has at most one conjunct in the body (i.e. conjunction is forbidden), and I, J two interpretations. If $I \sqsubseteq \alpha(J)$ then there exists a natural number h such that $\alpha(T_P^{lo}(I)) \sqsubseteq \alpha(T_P^{lo}{\uparrow_h}(J))$.

Proof
Suppose $I \sqsubseteq \alpha(J)$ and $\mathscr{A} \in \alpha(T_P^{lo}(I))$. Then there exists $\mathscr{A}' \in T_P^{lo}(I)$ s.t. $\mathscr{A} = \alpha(\mathscr{A}')$. By definition of T_P^{lo}, there exists a clause $H \circ\!\!- D \in P$ (the case for unit clauses is trivial) s.t. $\widehat{D} + \mathscr{C} \in I$ and $\mathscr{A}' = \widehat{H} + \mathscr{C}$. As $I \sqsubseteq \alpha(J)$, we also have $\widehat{D} + \mathscr{C} \in \alpha(J)$, which implies that there exists $\mathscr{K} \in J$ s.t. $\widehat{D} + \mathscr{C} = \alpha(\mathscr{K})$. Let $p = min\{n \mid \mathscr{K} \preccurlyeq (\widehat{D} + \mathscr{C})^n\}$ (it is immediate to prove that such a p exists), and let $\mathscr{M} = (\widehat{D} + \mathscr{C})^p$. We have that $\mathscr{K} \preccurlyeq \mathscr{M}$, therefore $\mathscr{M} \in J$ (because $\mathscr{K} \in J$ and J is upward-closed). Now, $\mathscr{M} = \widehat{D} + (\mathscr{C} + (\widehat{D} + \mathscr{C})^{p-1}) \in J$, and, by definition of T_P^{lo} ($H \circ\!\!- D \in P$), $\widehat{H} + \mathscr{C} + (\widehat{D} + \mathscr{C})^{p-1} \in T_P^{lo}(J)$. By repeatedly applying T_P^{lo} (the proof is by induction on

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

p) we get $(\widehat{H}+\mathscr{C})^p \in T_P^{lo}\uparrow_p (J)$. Therefore $\mathscr{A} = \alpha(\widehat{H}+\mathscr{C}) = \alpha((\widehat{H}+\mathscr{C})^p) \in \alpha(T_P^{lo}\uparrow_p (J))$. □

Proposition 12
Let P be an LO program in which every clause has at most one conjunct in the body. Then $\alpha(lfp(T_P^{lo})) = lfp(T_P^{\#})$.

Proof
By a simple induction, using Lemma 5, we have that for every k there exists h s.t. $T_P^{\#}\uparrow_k \sqsubseteq \alpha(T_P^{lo}\uparrow_h)$. Therefore $lfp(T_P^{\#}) \sqsubseteq \alpha(lfp(T_P^{lo}))$. □

The class of LO programs with one conjunct in the body is still very interesting. Below, we show how this result could be exploited to study reachability problems in Petri Nets.

11 Other applications: relation with Petri nets

As shown in the proof of Proposition 6, the class of propositional LO programs with one conjunct in the body is equivalent to VAS, i.e. to Petri nets. Intuitively, a multiset rewriting rule can be used to describe the effect of firing a Petri Net transition. For instance, the clause $a \parr b \parr b \circ\!\!- c \parr c$ can be interpreted as the Petri net transition that removes one token from place a, two tokens from place b, and adds two tokens to place c. As a consequence, a (possibly infinite) execution (sequence of goals $G_0, G_1, \ldots$) of a restricted LO program denotes an execution of the corresponding Petri Net. The initial goal G_0 can be viewed as the initial marking of the Petri Net. Consider now the fact $F \equiv c \circ\!\!- \top$, and let G_0 be the goal $a \parr a \parr b$. Then, the sequent $P \cup F \Rightarrow G_0$ is provable in LO if and only if there exists a reachable marking having *at least* one token in place c. In other words, the fact F can be used to implicitly represent an infinite set of markings (its upward closure) of the corresponding Petri net. Our bottom-up semantics can be use to effectively compute the set $Pre^*(F)$ (using the terminology of Abdulla *et al.* (1996)) of markings that can reach a marking in the denotation of F.

This idea can be used to verify *safety properties* of concurrent systems. A safety property S can be viewed as a set of *good* states (markings) of a given concurrent system (Petri net). The system satisfies the property if the set of states that are reachable from the initial state G_0 are all contained in S. Symmetrically, the set $\neg S$ represents the set of *bad* states. Thus, the systems can be proved correct by showing that $Pre^*(\neg S)$ does not contain the initial state G_0, i.e. by applying the bottom-up algorithm starting from a fact denoting $\neg S$. It is interesting to note that in many real examples $\neg S$ is indeed an upward closed set of states (e.g. the set of states where there are *at least two processes in the critical section* are the the typical bad states of a mutual exclusion algorithm). In general, the complexity of computing $Pre^*(F)$, for some F, can be very high. However, the results of Section 10 show that the fixpoint semantics of DLP can be used to approximate the set $Pre^*(F)$. Completeness implies that all properties that are preserved by the abstraction can be checked equivalently over the concrete and the abstract domain. In our setting

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

the kind of properties that satisfy this requirement can be informally characterized as 'at least one'-properties (e.g. *is there at least one token in place P in a reachable marking?*). This kind of properties can be used to check 'mutual exclusion' for a concurrent system represented via a Petri net. Suppose we want to prove that a system ensures mutual exclusion for two processes represented via a Petri net. Process p_i is in the critical section whenever a token is in a special place cs_i for $i : 1, 2$. Violations of mutual exclusion are expressed as the set of states with *at least* one token in place cs_1 and one token in state cs_2. Thus, the fixpoint semantics of the DLP program obtained as translation of the Petri Net (LO program) union the fact $cs_1 \vee cs_2$ is an abstraction of the set of backward reachable states. We obtain a full-test for mutual exclusion properties, whenever the initial states can be expressed again as *at least one* properties (i.e. whenever membership of the initial states in the set of abstract reachable states can be determined exactly).

12 Related work

Our work is inspired to the general decidability results for infinite-state systems based on the theory of well-quasi orderings given in Abdulla *et al.* (1996) and Finkel & Schnoebelen (2001). 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 Abdulla *et al.* (1996). Differently from Abdulla *et al.* (1996) and Finkel & Schnoebelen (2001), 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 Harland & Winikoff (1998), the authors present an abstract deductive system for the bottom-up evaluation of linear logic programs. The *left introduction* rules plus *weakening* and *cut* 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 $\mathrm{LO}_{\mathbf{1}}$, and they do not discuss computability issues for the resulting derivability relation.

Andreoli, Pareschi & Castagnetti (1997) 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 & Miller's (1969) *coverability tree* 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 $\mathscr{B}$, that has to be proved is subsumed by a node already visited, say $\mathscr{A}$, (i.e. $\mathscr{B} = \mathscr{A} + \mathscr{A}'$), then the part of proof tree between the two goals is replaced by a proof tree for $\mathscr{A} + (\mathscr{A}')^*$; $\mathscr{A} + (\mathscr{A}')^*$ is a finite representation of the union of $\mathscr{A}$ with the closure of $\mathscr{A}'$. 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

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

of contexts. This simplifies the computation as shown in Example 3 (we only need to test $\preccurlyeq$). Finally, differently from Andreoli *et al.* (1997), in this paper we have given also a formal semantics for the extension of LO with the constant **1**.

The partial evaluation scheme of Andreoli *et al.* (1997) 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 Kobayashi *et al.* (1995), where the authors present a type inference procedure that returns an approximation of the number of messages exchanged by HACL processes.

Cervesato (1995) shows how to encode Petri nets in LO, Lolli and Forum by exploiting the different features of these languages. We used some of these ideas to prove Proposition 6.

Finally, we have discussed the similarities between our semantics and the bottom-up semantics for Disjunctive Logic Programming of Minker, Rajasekar & Lobo (1991). 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 Minker *et al.* (1991) interpretations are collections of *sets* (disjunctions) of atomic formulas. Only minimal (w.r.t. 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 Minker *et al.* (1991) (collection of all minimal sets).

13 Conclusions and future work

In this paper we have defined a bottom-up semantics for the fragment of LinLog (Andreoli, 1992) consisting of the language LO (Andreoli & Pareschi, 1991) 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 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 (Hodas & Miller, 1994) and Lygon (Harland & Pym, 1994), and to study techniques for first-order formulation for all these languages. An extension of the present framework to the first-order case should also take into account the so-called *S-semantics* (Falaschi *et al.*, 1993; Bossi *et al.*, 1994), in order to model observables like *computed answer substitutions* and to cope with issues like *compositional* semantics. Concerning LO, we would also like to look at the connection with the so-called Chemical Abstract Machine metaphor (Andreoli *et al.*, 1993).

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Verification

In Delzanno & Podelski (1999), the authors show that properties of concurrent systems expressed in *temporal logic* can be defined in terms of fixpoint semantics of a logic program that encodes the transition system of a concurrent system. In Delzanno & Podelski (1999), 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 Delzanno & Podelski (1999) to concurrent systems in which synchronization is expressed at the logical level (see Section 11). The other way round, through the connection between semantics and verification, techniques used for infinite-state systems with integer variables (Delzanno & Podelski, 1999; Bultan *et al*., 1997; Henzinger *et al*., 1997) 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 Abdulla *et al.* (1996) and Finkel & Schnoebelen (2001) and provability in *sub-structural* logics like LO and *affine* linear logic (Kopylov, 1995). 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 Abdulla *et al.* (1996) and Finkel & Schnoebelen (2001)), i.e. a general notion of provability that ensures the termination of the bottom-up generation of valid formulas.

Relations between DLP and LO

We hope that our study will give rise to new ideas for the analysis of LO programs. As an example, it could be interesting to study weak notions of negation for LO that are based on the negation of DLP. Moreover, we can use DLP operational and fixpoint semantics to analyze Petri Nets, given that the abstraction is complete in this case. Finally, there are still some open questions concerning the relation between DLP and LO in the setting of abstract interpretation. In particular, we would like to study the notion of completeness for the general class of LO programs (we remark that the example in Bozzanno *et al.* (2000b) showing incompleteness was wrong). We would also like to analyze in more detail the connection between the notion of (full) completeness of the abstraction and proof-theoretic properties of provability in sub-structural logics, which has been only partly addressed in Bozzanno *et al.* (2000b).

Acknowledgements

The authors would like to thank Maurizio Gabbrielli for his support, Jean-Marc Andreoli for useful comments on the use of LO, and the anonymous reviewers for providing us suggestions and references that helped us to improve the presentation of the paper.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

References

Abdulla, P. A., Cerāns, K., Jonsson, B. and Tsay, Y.-K. (1996) General decidability theorems for infinite-state systems. *Proc. 11th Annual IEEE Int. Symposium on Logic in Computer Science (LICS'96)*, pp. 313–321. IEEE Computer Society Press.

Andreoli, J.-M. (1992) Logic Programming with Focusing Proofs in Linear Logic. *J. Logic and Computation*, **2**(3), 297–347.

Andreoli, J.-M. (1996) Coordination in LO. In: Andreoli, J.-M., Hankin, C. and Metayer, D. Le (eds.), *Coordination Programming: Mechanisms, Models and Semantics*, pp. 42–64. Imperial College Press.

Andreoli, J.-M. and Pareschi, R. (1991) Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, **9**(3-4), 445–473.

Andreoli, J.-M., Leth, L., Pareschi, R. and Thomsen, B. (1993) True concurrency semantics for a linear logic programming language with broadcast communication. In: Gaudel, M.-C. and Jouannaud, J.-P. (eds.), *Proc. Theory and Practice of Software Development, International Joint Conference CAAP/FASE (TAPSOFT'93): Lecture Notes in Computer Science 668*, pp. 182–198. Springer-Verlag.

Andreoli, J.-M., Pareschi, R. and Castagnetti, T. (1997) Static snalysis of linear logic programming. *New Generation Computing*, **15**(4), 449–481.

Bossi, A., Gabbrielli, M., Levi, G. and Martelli, M. (1994) The s-semantics approach: theory and applications. *J. Logic Programming*, **19-20**, 149–197.

Bozzano, M., Delzanno, G. and Martelli, M. (2000a) A bottom-up semantics for linear logic programs. In: Gabbrielli, M. and Pfenning, F. (eds.), *Proc. 2nd International Conference on Principles and Practice of Declarative Programming (PPDP'00)*, pp. 92–102. ACM Press.

Bozzano, M., Delzanno, G. and Martelli, M. (2000b) DLP as an abstraction of LO. On the relations between disjunctive and linear logic programming. *2000 Joint Conference on Declarative programming (APPIA-GULP-PRODE'00)*. Available from http://nutella.di.unipi.it/∼agp00/AccptAbstr.html.

Bultan, T., Gerber, R. and Pugh, W. (1997) Symbolic model checking of infinite state systems using Presburger arithmetics. In: O. Grumberg (ed.), *Proc. 9th International Conference on Computer Aided Verification (CAV'97): Lecture Notes in Computer Science 1254*, pp. 400–411. Springer-Verlag.

Cervesato, I. (1995) Petri nets and linear logic: a case study for logic programming. In: M. Alpuente and M. I. Sessa (eds.), *Proc. 1995 Joint Conference on Declarative Programming (GULP-PRODE'95)*, pp. 313–318. Palladio Press.

Cousot, P. and Cousot, R. (1977) Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fix-points. In: *Proc. 4th Symposium on Principles of Programming Languages (POPL'77)*, pp. 238–252. ACM Press.

Cousot, P. and Cousot, R. (1979) Systematic design of program analysis frameworks. *Proc. 6th Symposium on Principles of Programming Languages (POPL'79)*, pp. 269–282. ACM Press.

Cousot, P. and Halbwachs, N. (1978) Automatic discovery of linear restraints among variables of a program. *Proc. 5th Symposium on Principles of Programming Languages (POPL'78)*, pp. 84–96. ACM Press.

Delzanno, G. and Podelski, A. (1999) Model checking in CLP. In: Cleaveland, R. (ed.), *Proc. 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99): Lecture Notes in Computer Science 1579*, pp. 223–239. Springer-Verlag.

Dickson, L. E. (1913) Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Am. J. Mathematics*, **35**, 413–422.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press

Esparza, J. and Nielsen, M. (1994) Decidability issues for Petri nets – a survey. *J. Informatik Processing and Cybernetics*, **30**(3), 143–160.

Falaschi, M., Levi, G., Martelli, M. and Palamidessi, C. (1993) A model-theoretic reconstruction of the operational semantics of logic programs. *Information & Computation*, **103**(1), 86–113.

Finkel, A. and Schnoebelen, P. (2001) Well-structured transition systems everywhere! *Theoretical Computer Science*, **256**(1–2), 63–92.

Gabbrielli, M., Dore, M. G. and Levi, G. (1995) Observable semantics for Constraint Logic Programs. *J. Logic and Computation*, **5**(2), 133–171.

Giacobazzi, R. and Ranzato, F. (1997) Completeness in Abstract Interpretation: A Domain Perspective. In: M. Johnson (ed.), *Proc. 6th International Conference on Algebraic Methodology and Software Technology (AMAST'97): Lecture Notes in Computer Science 1349*, pp. 231–245. Springer-Verlag.

Girard, J. Y. (1987) Linear logic. *Theoretical Computer Science*, **50:1**, 1–102.

Harland, J. and Pym, D. J. (1994) A uniform proof-theoretic investigation of linear logic programming. *J. Logic and Computation*, **4**(2), 175–207.

Harland, J., & Winikoff, M. (1998). Making Logic Programs Reactive. *Pages 43–58 of: Proc. JICSLP'98 Workshop on Transactions and Change in Logic Databases (Dynamics'98)*.

Henzinger, T. A., Ho, P.-H. and Wong-Toi, H. (1997) HYTECH: A model checker for hybrid systems. In: Grumberg, O. (ed.), *Proc. 9th International Conference on Computer Aided Verification (CAV'97): Lecture Notes in Computer Science 1254*, pp. 460–463. Springer-Verlag.

Hodas, J. and Miller, D. (1994) Logic programming in a fragment of intuitionistic linear logic. *Information & Computation*, **110**(2), 327–365.

Jaffar, J. and Maher, M. J. (1994) Constraint logic programming: A survey. *J. Logic Programming*, **19-20**, 503–581.

Karp, R. M. and Miller, R. E. (1969) Parallel program schemata. *J. Computer & System Sciences*, **3**(2), 147–195.

Kobayashi, N. and Yonezawa, A. (1995) Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, **7**(2), 113–149.

Kobayashi, N., Nakade, M. and Yonezawa, A. (1995) Static analysis of communication for asynchronous concurrent programming languages. In: Mycroft, A. (ed.), *Proc. 2th International Symposium on Static Analysis (SAS'95): Lecture Notes in Computer Science 983*, pp. 225–242. Springer-Verlag.

Kopylov, A. .P. (1995) Decidability of linear affine logic. In: Kozen, D. (ed.), *Proc. 10th Annual IEEE Symposium on Logic in Computer Science (LICS'95)*, pp. 496–504. IEEE Computer Society Press.

Lloyd, J. .W. (1987) *Foundations of Logic Programming*. Springer-Verlag.

Miller, D. (1995) A survey of linear logic programming. *Computational Logic: The Newsletter of the European Network of Excellence in Computational Logic*, **2**(2), 63–67.

Miller, D. (1996) Forum: A multiple-conclusion specification logic. *Theoretical Computer Science*, **165**(1), 201–232.

Minker, J., Rajasekar, A. and Lobo, J. (1991) Theory of disjunctive logic programs. In: Lassez, J. L. and Plotkin, G. (eds.), *Computational Logic. Essays in Honor of Alan Robinson*, pp. 613–639. MIT Press.

https://doi.org/10.1017/S1471068402001254 Published online by Cambridge University Press