TLP **4** (5 & 6): 573–619, 2004. © 2004 Cambridge University Press
DOI: 10.1017/S1471068404002066 Printed in the United Kingdom

Model checking linear logic specifications

MARCO BOZZANO
ITC-IRST, Via Sommarive 18, Povo, 38050 Trento, Italy
(*e-mail:* bozzano@irst.itc.it)

Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Via Dodecaneso 35, 16146 Genova, Italy

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

Abstract

The overall goal of this paper is to investigate the theoretical foundations of *algorithmic* verification techniques for *first order linear logic* specifications. The fragment of linear logic we consider in this paper is based on the linear logic programming language called LO (Andreoli and Pareschi, 1990) enriched with *universally quantified goal formulas*. Although LO was originally introduced as a theoretical foundation for extensions of *logic programming* languages, it can also be viewed as a very general language to specify a wide range of *infinite-state concurrent systems* (Andreoli, 1992; Cervesato, 1995). Our approach is based on the relation between *backward reachability* and *provability* highlighted in our previous work on *propositional* LO programs (Bozzano *et al*., 2002). Following this line of research, we define here a general framework for the *bottom-up* evaluation of *first order* linear logic specifications. The evaluation procedure is based on an effective fixpoint operator working on a symbolic representation of infinite collections of first order linear logic formulas. The theory of well quasi-orderings (Abdulla *et al*., 1996; Finkel and Schnoebelen, 2001) can be used to provide sufficient conditions for the termination of the evaluation of non trivial fragments of first order linear logic.

KEYWORDS: linear logic, fixpoint semantics, bottom-up evaluation

1 Introduction

The algorithmic techniques for the analysis of Petri Nets are based on very well consolidated theoretical foundations (Esparza and Melzer, 2000; Karp and Miller, 1969; Mayr, 1984; Esparza *et al*., 1999; Finkel, 1993; Silva *et al*., 1998). However, several interesting problems, e.g. the *coverability* problem, become undecidable when considering specification languages more expressive than basic Petri Nets. In this setting, *validation* of complex specifications is often performed through *simulation* and *testing*, i.e. by "executing" the specification looking for design errors, e.g. as in the methodology based on the construction of the reachability graph of Colored

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Petri Nets (Jensen, 1997). To study algorithmic techniques for the analysis of a vast range of concurrency models it is important to find a uniform framework to reason about their characteristic features.

In our approach we adopt *linear logic* (Girard, 1987) as a unified logical framework for concurrency. Linear logic provides a logical characterization of concepts and mechanisms peculiar of concurrency like *locality*, *recursion*, and *non determinism* in the definition of a process (Andreoli and Pareschi, 1990; Kobayashi and Yonezawa, 1995; Martí-Oliet and Meseguer, 1991); communication via *synchronization* and *value passing* (Cervesato, 1995; Miller, 1993); *internal state and updates to its current value* (Andreoli and Pareschi, 1990; Miller, 1996); and *generation of fresh names* (Cervesato *et al*., 1999; Miller, 1993). *Provability* in fragments of linear logic can be used as a formal tool to reason about behavioral aspects of the *concurrent systems* (Bozzano *et al*., 2002; McDowell *et al*., 1996).

The overall goal of this paper is to investigate the theoretical foundations of *algorithmic* verification techniques for specifications based on *first order linear logic*. The fragment we consider in this paper is based on the linear logic programming language called LO (Andreoli and Pareschi, 1990) enriched with *universally quantified goal formulas*. Apart from being a logic programming language, the appealing feature of LO is that it can also be viewed as a rich *specification* language for concurrent systems:

- Specification languages like Petri Nets and multiset rewriting over first order atomic formulas can be naturally embedded into *propositional LO* (e.g. see Cervesato (1995) and Bozzano *et al*. (2002)).
- *First order LO specifications* can be used to specify the internal state of processes with structured data represented as terms, thus enlarging the class of systems that can be formally specified in the logic. In this context universal quantification in goal formulas has several interesting interpretations: it can be viewed either as a sort of *hiding* operator in the style of π-calculus (Miller, 1993), or as a mechanism to generate *fresh names* as in Cervesato *et al*. (1999).

Before discussing in more details the technical contributions of our work, we will briefly illustrate the connection between Petri Nets and linear logic, and between reachability and provability in the corresponding formal settings. The bridge between the two paradigms is the *proofs as computations* interpretation of linear logic proposed in Andreoli (1992) and Miller (12996).

Linear Logic and Concurrency A Petri Net can be represented by means of a multiset-rewriting system over a finite alphabet, say $p, q, r, \ldots$, of *place* names. One possible way of expressing multiset rewrite rules in linear logic is based on the following idea. The connective $\parr$ (multiplicative disjunction) is interpreted as a multiset constructor, whereas the connective $\multimap$ (reversed linear implication) is interpreted as the rewrite relation. Both connectives are allowed in the LO fragment. For instance, as shown in Cervesato (1995) the LO clause

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

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

can be viewed as a Petri Net *transition* that removes a token from places p and q and puts two tokens in place p, one in q, and one in t. According to the proofs as computations interpretation (Andreoli, 1992), a *top-down* derivation in linear logic consists of a goal-directed sequence of rule applications. If we look at the initial goal as a multiset of atomic formulas (places) representing the initial marking of a Petri Net, then each application of an LO clause like the one illustrated above (*backchaining* in the terminology of Andreoli (1992)) simulates the firing of a Petri Net transition at the corresponding marking. Furthermore, the overall top-down derivation corresponds to one of the possible executions of the net, leading from the initial marking to one of the target states.

Thanks to the presence of other connectives, LO supports more sophisticated mechanisms than the ones available in simple Petri Nets. For instance, Andreoli and Pareschi (1990) use LO clauses with occurrences of $\parr$ and $\&$ (additive conjunction) in their *body* to express what they called *external* and *internal* concurrency. Additive conjunction can be used, in fact, to simulate independent threads of execution running in parallel.

In our previous work (Bozzano *et al.*, 2002), we made a first attempt to connect techniques used for the validation of Petri Nets with evaluation strategies of LO programs. Specifically, in Bozzano *et al.* (2002) we defined an effective procedure to compute the set of linear logic *goals* (multisets of atomic formulas) that are consequences of a given propositional program, i.e. a "bottom-up" evaluation procedure for *propositional* LO programs. Our construction is based on the *backward reachability* algorithm of Abdulla *et al.* (1996) used to decide the so called *control state reachability problem* of Petri Nets (i.e. the problem of deciding if a given set of *upward closed* configurations are reachable from an initial one). The algorithm works as follows. Starting from a set of *target states*, the algorithm computes symbolically the transitive closure of the *predecessor* relation (i.e. the transition relation read backwards) of the Petri Net taken into consideration. The algorithm is used to check safety properties: if the algorithm is executed starting from the set of *unsafe states*, then the corresponding safety property holds if and only if the initial marking is not in the resulting fixpoint.

To illustrate the connection between backward reachability for Petri Nets and provability in LO, we first observe that LO program clauses of the form

$$p \parr q \parr q \multimap \top$$

succeed in any context containing *at least* one occurrence of p and two occurrences of q. In other words they can be used to symbolically represent sets of markings that are closed upwards with respect to the multiset inclusion relation. Now, suppose we represent a Petri Net via an LO program P and the set of *target states* using a collection T of LO program clauses with $\top$ in the body. Then, the set of *facts* (i.e. multisets of atomic formulas) that are *logical consequences* of the LO program $P \cup T$ will represent the set of markings that are *backward reachable* from the target states.

The algorithm we presented in Bozzano *et al.* (2002) is based on this idea, and it extends the backward reachability algorithm for Petri Nets of Abdulla *et al.* (1996)

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

to the more general case of propositional LO programs (i.e. with nested conjunctive and disjunctive goals).

First Order Linear Logic By lifting the logic language to *first order*, the resulting specification language becomes much more interesting and flexible than basic Petri Nets. In the extended setting, the logic representation of processes can be enriched with a notion of *internal state* and with communication mechanisms in which *values* can be passed between different processes. As an example, the following LO clause

$$idle(Y) \parr p(alice, wait, stored(Y)) \multimap p(alice, use, stored(Y))$$

can be interpreted as a transaction of a protocol during which the *process* named Alice (currently knowing Y) synchronizes with a monitor controlling the resource Y, checks that the monitor is *idle* and then enters the critical section in which she uses the resource Y. By instantiating the free variables occurring in such a rule, we obtain a family of transition rules that depend on the domain used to define the content of messages. In this setting the universal quantification in goal formulas can be used to generate *fresh values*, as in the following rule:

$$init \multimap \forall x.\ idle(x) \parr init$$

Intuitively, the demon process *init* creates new resources labeled with fresh identifiers.

The above illustrated connection between *provability* and *reachability* immediately gives us a well-founded manner of extending the algorithmic techniques used for the analysis of Petri Nets to the general case of first order linear logic specifications.

Our Contribution The conceptual and technical contributions of our work can be summarized as follows.

(1) Combining ideas coming from the semantics of logic programming (Bossi *et al.*, 1994; Falaschi *et al.*, 1993) and from symbolic model checking for infinite state systems (Abdulla *et al.*, 1996; Finkel and Schnoebelen, 2001), in this paper we present the theoretical foundations for the definition of a procedure for the *bottom-up* evaluation of first order LO programs with universally quantified goals. By working in the general setting of linear logic, we obtain a framework that can be applied to other specification languages for concurrent systems like *multiset rewriting over first order atomic formulas* (Cervesato *et al.*, 1999). The bottom-up evaluation procedure can also be viewed as a *fixpoint* semantics that allows us to compute the set of all goals that are linear logical consequences of a given (extended) LO program. The fixpoint semantics is based on an *effective* fixpoint operator and on a *symbolic* and *finite* representation of an *infinite* collection of first order provable LO goals. As previously mentioned, the possible infiniteness of the set of provable goals is due to LO program clauses with the constant $\top$, which represent sets of goals which are upward-closed with respect to the multiset inclusion relation. The symbolic representation is therefore crucial when trying to prove properties of infinite systems like *parameterized* systems, i.e. systems in which the number of individual processes is left as a parameter of the specification (e.g. mutual exclusion protocols for *multi-agent* systems (Bozzano, 2002)). Intuitively, such

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

a representation is obtained by restricting our attention to logical consequences represented via multisets of first order atomic formulas. As an example, the formula

$$p(A, use, stored(X)) \ \parr \ p(B, use, stored(X)) \ \multimap \ \top$$

can be used to denote all multisets of *ground* atomic formulas *containing* an instance of the clause head. As the constant $\top$ is provable in any context, in the previous example we obtain a *symbolic representation* of the infinite set of *unsafe states* generated by the following minimal violation of mutual exclusion for a generic resource represented via the shared variable X: *at least two different processes are in their critical section using a shared resource.*

(2) Besides the connection with verification of concurrent systems, the new fixpoint semantics for first order LO programs represents an alternative to the traditional top-down execution of linear logic programs studied in the literature (Andreoli, 1992). Thus, also from the point-of-view of logic programming, we extend the applicability of our previous work (Bozzano *et al.*, 2002) (that was restricted to the propositional case) towards more interesting classes of linear logic programs.

(3) The termination of the fixpoint computation cannot be guaranteed in general; first order LO programs are in fact Turing complete. However, we present here sufficient conditions under which we can compute a *symbolic representation of all logical consequences* of a non trivial first order fragment of LO with universal quantification in goal formulas. As a direct consequence of this result, we obtain that provability is decidable in the considered fragment. To our knowledge, this result uncovers a new decidable fragment of first order linear logic. The fragment taken into consideration is not only interesting from a theoretical point of view, but also as a possible abstract model for "processes" with identifiers or local values.

Though the emphasis of this work is on the theoretical grounds of our method, we will illustrate the practical use of our framework with the help of a verification problem for a *mutual exclusion protocol* defined for a concurrent system which is parametric in the number of *clients*, *resources*, and related *monitors*. Other practical applications of this method are currently under investigation. Preliminary results in this direction are shown in the PhD thesis of Marco Bozzano (Bozzano, 2002).

Finally, we remark that a very preliminary version of this work appeared in the proceedings of FLOPS 2001 (Bozzano *et al.*, 2001).

1.1 Outline of the paper

The terminology and some notations used in the paper are presented in Appendix A. For lack of space, the proofs of some lemmas have been omitted, the reader is referred to the extended version of this paper (Bozzano *et al.*, 2003). In Section 2, we will discuss related works. In Section 3 we will recall the main definitions of the fragment LO of Andreoli and Pareschi (1990), presented here with universal quantification in goal formulas. In order to illustrate the use of LO as a specification

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

logic for concurrent systems, in the same section we will briefly describe how *multiset rewriting* (extended with quantification) can be embedded into LO. This connection represents a natural entry point into the world of concurrency. In fact, the relationship between multiset rewriting, (Colored) Petri Nets, and process calculi has been extensively studied in the literature (Cervesato, 1995; Farwer, 1999; Farwer, 2000; Meseguer, 1992; Martí-Oliet and Meseguer, 1991). Finally, we will present an example of use of LO as a specification language for concurrent systems, and discuss the relationships between (bottom-up) LO provability and verification techniques based on (infinite-state) model checking. In Section 4, we will introduce a *non effective* fixpoint semantics for linear logic programs. To simplify the manipulation of non ground terms, we will first lift the top-down (proof theoretical) semantics of LO to the non ground level, by introducing a new proof system in which sequents may have formulas with free variables. In Section 5, we will introduce a general framework for the bottom-up evaluation of LO programs. The bottom-up procedure is based on a finite representation of infinite sets of logical consequences, and on an effective fixpoint operator working on sets of symbolic representations. The bottom-up procedure can be seen as a symbolic version of the semantics presented in Section 4. The reason for introducing two different semantic definitions is to ease the proof of soundness and completeness, which is split into the proof of equivalence of the effective semantics with respect to the non-effective one, and the proof of equivalence of the non-effective semantics with respect to the operational one. In Section 6, we will investigate sufficient conditions for the termination of the bottom-up evaluation. In Section 7, we will discuss the possible application of the resulting method as a verification procedure for *infinite-state parameterized systems*. In Section 8, we will address possible future directions of research. In Section 9, we will address some conclusions.

2 Related work

To our knowledge, our work is the first attempt to connect algorithmic techniques used in symbolic model checking with declarative and operational aspects of *first order* linear logic programming. In Bozzano *et al.* (2002), we have considered the relation between *propositional* LO and Petri Nets. Specifically, in Bozzano *et al.* (2002) we have shown that the bottom-up semantics is computable for propositional LO programs (because of the relationship of this problem with the coverability problem of Petri Nets). Furthermore, in Bozzano *et al.* (2002) we have shown that the bottom-up evaluation of propositional LO programs enriched with the constant **1** is not computable in a finite number of steps (otherwise one could decide the equivalence problem for Petri Nets).

We point out here that an original contribution of the paper consists in extending the construction we used for proving the computability of the bottom-up construction of propositional LO programs to *first order* LO specifications. This way, we have established a link with more complex models of concurrency. Clearly, in the first order case provability becomes undecidable. In the paper we present a non trivial special case of first order LO programs in which the bottom-up semantics is still

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

computable. Extending the bottom-up evaluation to LO programs enriched with the constant **1** is a possible future direction of research (see Section 8 for a discussion).

Harland and Winikoff (1998) presented an abstract deductive system for bottom-up evaluation of linear logic programs. The left introduction plus weakening and cut rules are used to compute the logical consequences of a given formula. Though the framework is given for a more general fragment than LO, it does not provide an *effective* procedure to evaluate programs. Andreoli *et al.* (1997) define an improved *top-down* strategy for *propositional* LO based on the Karp-Miller's covering graph of Petri Nets, i.e. a *forward* exploration with accelerations.

The relation between Rewriting, (Colored) Petri Nets and Linear Logic has been investigated in previous works (Cervesato, 1994; Cervesato, 1995; Engberg and Winskel, 1990; Meseguer, 1992; Martí-Oliet and Meseguer, 1991). Our point-of-view is based on the proofs as computations metaphor (Andreoli and Pareschi, 1990; Andreoli, 1992; Miller, 1996), whereas our connection with models for concurrency is inspired by other works in this field (Cervesato, 1994; Cervesato, 1995; Delzanno and Martelli, 2001; Kobayashi and Yonezawa, 1994; Miller, 1993; Miller, 1996). As an example, Cervesato (1994, 1995) shows how to encode Petri Nets in different fragments of linear logic like LO, Lolli (Hodas and Miller, 1990), and Forum (Miller, 1996) exploiting the different features of these languages. Algorithmic aspects for verification of properties of the resulting linear logic specifications are not considered in the works mentioned above. Farwer (1999, 2000) presents a possible encoding of Colored Petri Nets in Linear Logic and proposes a combination of the two formalisms that could be used to model object systems.

The problem of the *decidability* of provability in fragments of linear logic has been investigated in several works in recent years (Lincoln, 1995; Lincoln *et al.*, 1992; Lincoln and Scedrov, 1994). Specifically, Kopylov (1995) has shown that the full *propositional* linear *affine* logic containing all the multiplicatives, additives, exponentials, and constants is decidable. Affine logic can be viewed as linear logic with the *weakening rule*. Propositional LO belongs to such a sub-structural logic. Provability in full first order linear logic is undecidable as shown by Girard's translation of first order logic into first order linear logic (Girard, 1987). The same holds for first order affine logic (Girard's encoding can also be viewed as an encoding into affine logic (Lincoln, 1995)). First order linear logic *without modalities*, i.e. without the possibility of re-using formulas, is decidable (Lincoln and Scedrov, 1994). Cervesato *et al.* (1999) use a formalism based on multiset-rewriting and existential quantification that can be embedded into our fragment of linear logic to specify protocol rules and actions of intruders. In Durgin *et al.* (1999), it is shown that reachability in multiset rewriting with existential quantification is undecidable by a reduction from Datalog with quantification in goal formulas. The fragment they consider however is much more general than the *monadic* fragment of $LO_{\forall}$. Monadic $LO_{\forall}$ can be viewed as a fragment of first order linear affine logic with restricted occurrences of the exponentials (program clauses are re-usable) and severe restrictions on the form of atomic formulas. We are not aware of previous results on similar fragments.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

3 The logic programming language LO

LO (Andreoli and Pareschi, 1991) is a logic programming language based on a fragment of LinLog (Andreoli, 1992). Its mathematical foundations lie on a proof-theoretical presentation of a fragment of linear logic defined over the linear connectives $\multimap$ (*linear implication*, we use the reversed notation $H \circ\!\!- G$ for $G \multimap H$), & (*additive conjunction*), $\parr$ (*multiplicative disjunction*), and the constant $\top$ (*additive identity*). In this section we present the proof-theoretical semantics, corresponding to the usual *top-down* operational semantics for traditional logic programming languages, for an extension of LO. First of all, we consider a slight extension of LO which admits the constant $\bot$ in goals and clause heads. More importantly, we allow the universal quantifier to appear, possibly nested, in goals. This extension is inspired by *multiset rewriting with universal quantification* (Cervesato *et al.*, 1999). The resulting language will be called $\mathrm{LO}_\forall$ hereafter. Following Andreoli and Pareschi (1991), we give the following definitions.

Definition 3.1 (*Atomic Formulas*)
Let Σ be a signature with predicates including a set of constant and function symbols $\mathscr{L}$ and a set of predicate symbols $\mathscr{P}$, and let $\mathscr{V}$ be a denumerable set of variables. An atomic formula over Σ and $\mathscr{V}$ has the form $p(t_1, \ldots, t_n)$ (with $n \geqslant 0$), where $p \in \mathscr{P}$ and $t_1, \ldots, t_n$ are (non ground) terms in $T_\Sigma^{\mathscr{V}}$. We denote the set of such atomic formulas as $A_\Sigma^{\mathscr{V}}$.

We are now ready to define $\mathrm{LO}_\forall$ programs. The class of **D**-formulas correspond to multiple-headed program clauses, whereas **G**-formulas correspond to *goals* to be evaluated in a given program.

Definition 3.2 ($LO_\forall$ *programs*)
Let Σ be a signature with predicates and $\mathscr{V}$ a denumerable set of variables. The classes of **G**-formulas (goal formulas), **H**-formulas (head formulas), and **D**-formulas (program clauses) over Σ and $\mathscr{V}$ are defined by the following grammar:

$$\begin{aligned}
\mathbf{G} &::= \mathbf{G} \parr \mathbf{G} \mid \mathbf{G} \,\&\, \mathbf{G} \mid \forall x.\mathbf{G} \mid \mathbf{A} \mid \top \mid \bot \\
\mathbf{H} &::= \mathbf{A} \parr \ldots \parr \mathbf{A} \mid \bot \\
\mathbf{D} &::= \mathbf{H} \circ\!\!- \mathbf{G} \mid \mathbf{D} \,\&\, \mathbf{D} \mid \forall x.\mathbf{D}
\end{aligned}$$

where **A** stands for an atomic formula over Σ and $\mathscr{V}$. An $\mathrm{LO}_\forall$ program over Σ and $\mathscr{V}$ is a **D**-formula over Σ and $\mathscr{V}$. A multiset of goal formulas will be called a *context* hereafter.

Remark 3.3
Given an $\mathrm{LO}_\forall$ program P, in the rest of the paper we often find it convenient to view P as the *set* of clauses $D_1, \ldots, D_n$. Every *program clause* D_i has the form $\forall(H \circ\!\!- G)$ standing for $\forall x_1 \ldots x_k.(H \circ\!\!- G)$, where $FV(H \circ\!\!- G) = \{x_1, \ldots, x_k\}$.
Formally, this is justified by the following logical equivalences (Girard, 1987):

$$\begin{aligned}
!(D_1 \,\&\, D_2) &\equiv\ !D_1 \otimes\ !D_2 \\
\forall x.(D_1 \,\&\, D_2) &\equiv\ \forall x.D_1 \ \&\ \forall x.D_2
\end{aligned}$$

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

For the sake of simplicity, in the following we usually omit the universal quantifier in **D**-formulas, i.e. we consider free variables as being *implicitly* universally quantified.

Definition 3.4 (*$LO_\forall$ Sequents*)
Let Σ be a signature with predicates and $\mathscr{V}$ a denumerable set of variables. An $LO_\forall$ sequent has the form $P \vdash_{\Sigma'} G_1, \ldots, G_k$, where P is an $LO_\forall$ program over Σ and $\mathscr{V}$, $G_1, \ldots, G_k$ is a context (i.e. a multiset of goals) over Σ and $\mathscr{V}$, and Σ' is a signature such that $\Sigma \subseteq \Sigma'$.

According to Remark 3.3, structural rules (*exchange*, *weakening* and *contraction*) are allowed on the left-hand side, while on the right-hand side only the rule of exchange is allowed (for the fragment under consideration, it turns out that the rule of weakening is admissible, while contraction is forbidden). We now define provability in $LO_\forall$.

Definition 3.5 (*Ground Instances*)
Let Σ be a signature with predicates and $\mathscr{V}$ a denumerable set of variables. Given an $LO_\forall$ program P over Σ and $\mathscr{V}$, the set of ground instances of P, denoted $Gnd(P)$, is defined as follows: $Gnd(P) \stackrel{\text{def}}{=} \{(H \circ\!\!- G)\,\theta \mid \forall\,(H \circ\!\!- G) \in P\}$, where θ is a grounding substitution for $H \circ\!\!- G$ (i.e. it maps variables in $FV(H \circ\!\!- G)$ to ground terms in T_Σ).

The execution of a multiset of **G**-formulas $G_1, \ldots, G_k$ in P corresponds to a *goal-driven* proof for the sequent $P \vdash_\Sigma G_1, \ldots, G_k$. According to this view, the operational semantics of $LO_\forall$ is given via the *uniform* (*focusing*) (Andreoli, 1992) proof system presented in Figure 1, where P is a set of clauses, $\mathscr{A}$ is a multiset of atomic formulas, and Δ is a multiset of **G**-formulas. We have used the notation $\widehat{H}$, where H is a linear disjunction of atomic formulas $A_1 \parr \ldots \parr A_n$, to denote the multiset $A_1, \ldots, A_n$ (by convention, $\widehat{\bot} = \epsilon$, where ϵ is the empty multiset).

Definition 3.6 (*$LO_\forall$ provability*)
Let Σ be a signature with predicates and $\mathscr{V}$ a denumerable set of variables. Given an $LO_\forall$ program P and a goal G, over Σ and $\mathscr{V}$, we say that G is provable from P if there exists a proof tree, built over the proof system of Figure 1, with root $P \vdash_\Sigma G$, and such that every branch is terminated with an instance of the $\top_r$ axiom.

The concept of *uniformity* applied to LO requires that the right rules $\top_r$, $\parr_r$, $\&_r$, $\bot_r$, $\forall_r$ have priority over *bc*, i.e. *bc* is applied only when the right-hand side of a sequent is a multiset of *atomic* formulas (as suggested by the notation $\mathscr{A}$ in Figure 1). The proof system of 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). Rule *bc* is analogous to a backchaining (resolution) step in traditional logic programming languages. Note that according to the concept of resolution explained above, *bc* can be executed only if the right-hand side of the current $LO_\forall$ sequent consists of atomic formulas. As an instance of rule *bc*, we get

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

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

$$\frac{P \vdash_\Sigma \Delta}{P \vdash_\Sigma \bot, \Delta}\ \bot_r \qquad \frac{P \vdash_{\Sigma, c} G[c/x], \Delta}{P \vdash_\Sigma \forall x.G, \Delta}\ \forall_r\ (c \notin \Sigma) \qquad \frac{P \vdash_\Sigma G, \mathscr{A}}{P \vdash_\Sigma \widehat{H}, \mathscr{A}}\ bc\ \ (H \multimap G\ \in Gnd(P))$$

Fig. 1. A proof system for $\mathrm{LO}_\forall$.

the following proof fragment, which deals with the case of clauses with empty head:

$$\frac{\begin{array}{c}\vdots\\ P \vdash_\Sigma \mathscr{A}, G\end{array}}{P \vdash_\Sigma \mathscr{A}}\ bc$$

$$provided\ \bot \multimap G \in Gnd(P)$$

Given that clauses with empty head are always applicable in atomic *contexts*, the degree of non-determinism they introduce in proof search is usually considered unacceptable (Miller, 1996) and in particular they are forbidden in the original presentation of LO (Andreoli and Pareschi, 1991). However, the computational model we are interested in, i.e. bottom-up evaluation, does not suffer this drawback. Clauses with empty head often allow more flexible specifications.

LO clauses having the form $H \multimap \bot$ simply remove the resources associated with H from the right-hand side of the current sequent (H is rewritten into the empty multiset). On the contrary, LO clauses having the form $H \multimap \top$ can be viewed as *termination* rules. In fact, when a backchaining step over such a clause is possible, we get a *successful* (branch of a) computation, independently of the current *context* $\mathscr{A}$, as shown in the following proof scheme:

$$\frac{\dfrac{}{P \vdash_\Sigma \top, \mathscr{A}}\ \top_r}{P \vdash_\Sigma \widehat{H}, \mathscr{A}}\ bc$$

$$provided\ H \multimap \top \in Gnd(P)$$

This observation is formally stated in the following proposition (we recall that $\preccurlyeq$ is the multiset inclusion relation).

Proposition 1 (*Admissibility of the Weakening Rule*)
Given an $\mathrm{LO}_\forall$ program P and two multisets of goals Δ, Δ' such that $\Delta \preccurlyeq \Delta'$, if $P \vdash_\Sigma \Delta$ then $P \vdash_\Sigma \Delta'$.

Proof
By simple induction on the structure of $\mathrm{LO}_\forall$ proofs. □

Admissibility of the weakening rule makes $\mathrm{LO}_\forall$ an *affine* fragment of linear logic (Kopylov, 1995). Note that all structural rules are admissible on the left hand side (i.e. on the program part) of $\mathrm{LO}_\forall$ sequents.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$$\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma,c} \top}\,\top_r}{P \vdash_{\Sigma,c} p(f(c)), q(f(c))}\,bc^{(4)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma,c} \top, s(c)}\,\top_r}{P \vdash_{\Sigma,c} p(f(c)), q(f(c)), s(c)}\,bc^{(4)}}{P \vdash_{\Sigma,c} p(f(c)), q(f(c)) \parr s(c)}\,\parr_r}{P \vdash_{\Sigma,c} p(f(c)), r(c)}\,bc^{(1)}}{P \vdash_{\Sigma,c} p(f(c)), q(f(c)) \,\&\, r(c)}\,\&_r$$
$$\dfrac{P \vdash_{\Sigma,c} p(f(c))}{P \vdash_{\Sigma} \forall x.p(f(x))}\,\forall_r$$
$$\dfrac{}{P \vdash_{\Sigma} s(a)}\,bc^{(2)}$$

Fig. 2. An example of $\mathrm{LO}_\forall$ proof.

Finally, rule $\forall_r$ can be used to *dynamically* introduce new *names* during the computation. The initial signature Σ must contain at least the constant, function, and predicate symbols of a given program P, and it can dynamically grow thanks to rule $\forall_r$.

Remark 3.7

Particular attention must be paid to the constants introduced in a derivation. They cannot be extruded from the scope of the corresponding universal quantifier. For this reason, every time rule $\forall_r$ is applied, a new constant c is added to the current signature, and the resulting goal is proved in the new signature. The idea is that all terms appearing on the right-hand side of a sequent are implicitly assumed to range over the relevant signature. This behavior is standard in logic programming languages (Miller *et al.*, 1991).

Example 3.8

Let Σ be a signature with a constant symbol a, a function symbol f and predicate symbols p, q, r, s. Let $\mathscr{V}$ be a denumerable set of variables, and $u, v, w, \ldots \in \mathscr{V}$. Let P be the program

1. $r(w) \circ\!\!- q(f(w)) \parr s(w)$
2. $s(z) \circ\!\!- \forall x.p(f(x))$
3. $\bot \circ\!\!- q(u) \,\&\, r(v)$
4. $p(x) \parr q(x) \circ\!\!- \top$

The goal $s(a)$ is provable from P. The corresponding proof is shown in Figure 2 (where we have denoted by $bc^{(i)}$ the application of the backchaining rule over clause number i of P). Note that the notion of *ground instance* is now relative to the current signature. For instance, backchaining over clause 3 is possible because the corresponding signature contains the constant c (generated one level below by the $\forall_r$ rule), and therefore $\bot \circ\!\!- q(f(c)) \,\&\, r(c)$ is a valid instance of clause 3. □

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

3.1 Simulating multiset rewriting over first order atoms

In this section we will focus our attention on the relationship between *multiset rewriting over first order atoms* and *first order LO theories*. We will conclude by showing how enriching logic theories with universal quantification can provide a way to generate *new values*.

The connection between multiset rewriting systems over (first order) atomic formulas and (first order) LO theories has been studied by Cervesato (1994) and Cervesato *et al.* (1999). Cervesato (1994) presents different possible encodings of multiset rewriting (without function symbols) in linear logic. Specifically, he first presents an encoding in the multiplicative fragment of intuitionistic linear logic (MILL), where multiplicative conjunction $\otimes$ ("tensor") and linear implication are used as multiset constructor and rewrite relation, respectively. As an example, the formula $p \otimes q \multimap r \otimes s$ represents a rewrite rule in which p and q are rewritten into r and s ($\otimes$ denotes the "tensor").

As highlighted in Remark 5.12 of Cervesato (1994), an equivalent encoding can be given by choosing a fragment of classical linear logic contained in LO in which multiplicative disjunction and reverse linear implication are used as multiset constructor and rewrite relation, respectively. As an example, the formula $p \parr q \circ\!\!- r \parr s$ represents a rewrite rule in which p and q are rewritten into r and s. This is the encoding we will adopt in our work.

The duality of the two encodings is a consequence of the following property: $(p \otimes q \multimap r \otimes s)^{\perp} \equiv p^{\perp} \parr q^{\perp} \circ\!\!- r^{\perp} \parr s^{\perp}$, where $a^{\perp}$ is the linear logic negation of a. Furthermore, it depends on the way proofs are interpreted as computations, i.e. on whether "rewrite rules" are encoded as formulas that occur on the left- or on the right-hand side of a sequent.

Cervesato (1994, Section 5.2.2) also presents an encoding of Petri Nets in LO that allows one to simulate the execution of a net using an LO top-down derivation of the resulting program. In Section 5 of Cervesato *et al.*, 1999) the encoding of multiset rewriting over first order atomic formulas (MSR) is extended to first order MILL with existential quantifiers. Thanks to its logical nature, the duality with the first order fragment of LO still holds.

To illustrate the main ideas behind the interpretation of LO as multiset rewriting, let us first define the following class of LO formulas.

Definition 3.9
We call *LO rewrite rule* any LO formula having the following form

$$\forall(A_1 \parr \ldots \parr A_n \ \circ\!\!- \ B_1 \parr \ldots \parr B_m)$$

where $A_1, \ldots, A_n$ and $B_1, \ldots, B_m$ are atomic formulas over Σ and $\mathscr{V}$.

As usual, the notation $\forall(H \circ\!\!- G)$ stands for the universal quantification of clause $H \circ\!\!- G$ over its free variables.

LO formulas having the form depicted above can be interpreted as *multiset rewriting rules* in which rewriting can be performed only at the level of atomic formulas as in the MSR framework defined in Cervesato *et al.* (1999).

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

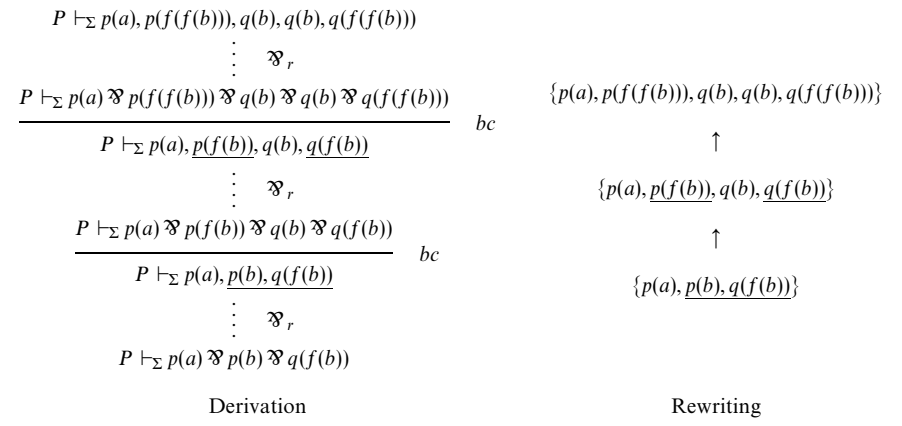


Fig. 3. Multiset rewriting over first order atomic formulas as LO proof construction.

Specifically, let P be a set of LO rewrite rules (as in Definition 3.9). Now, consider a goal formula G having the form $C_1 \parr \ldots \parr C_k$ where $C_1, \ldots, C_k$ are *ground* atomic formulas over Σ. It is easy to verify that any derivation starting from $P \vdash_\Sigma G$ and built using LO proof rules amounts to a sequence of multisets rewriting steps, where $\parr$ is interpreted as multiset constructor.

Example 3.10

Let Σ be a signature with two constant symbols a and b, one function symbol f and two predicate symbols p, q. Let $\mathscr{V}$ be a denumerable set of variables and $x, y \in \mathscr{V}$. Let P consists of the LO clause

$$\forall x, y.\, p(x) \parr q(f(y)) \circ\!\!- p(f(x)) \parr q(y) \parr q(f(x))$$

and $G = p(a) \parr p(b) \parr q(f(b))$. Figure 3 shows one possible sequence of applications of the above clause that starts from the sequent $P \vdash_\Sigma G$ (we have underlined atomic formulas selected in the application of the *bc* rule). □

From the previous example, we can observe the following properties. All derivations built using *LO rewrite rules* of Def. 3.9 consist of applications of $\parr_r$ and *bc*. Thus, they have no branching (all derivations form a single line). The combination of a sequence of applications of the $\parr_r$ rule and of the backchaining rule has the following effect: the head of a ground instance of a rule in P is matched against a sub-multiset in the current goal; the selected multiset is replaced by the body of the rule. Clearly, this property allows us to *simulate* multiset rewriting over first order atomic formulas by using LO rewrite rules.

Now, let F_1 be the clause

$$p(a) \parr p(f(f(b))) \parr q(b) \parr q(b) \parr q(f(f(b))) \circ\!\!- \top$$

If we enrich P with F_1, then we can transform the partial derivation of Figure 3 into an LO proof as shown below (where δ stands for the derivation fragment of

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Figure 3):

$$\dfrac{\dfrac{}{P \vdash_\Sigma \top}\ \top_r}{\delta}\ bc$$

It is important to note that the same effect can be achieved by adding any formula with $\top$ that contains a *sub-multiset* of the right-hand side of the last sequent in the derivation of Figure 3. As an example, let F_2 be the formula

$$p(a) \parr q(b) \circ\!\!- \top$$

If we enrich P with F_2, then we can transform the partial derivation of Figure 3 into an LO proof as shown below (again, δ stands for the derivation fragment of Figure 3):

$$\dfrac{\dfrac{}{P \vdash_\Sigma \top, p(f(f(b))), q(b), q(f(f(b)))}\ \top_r}{\delta}\ bc$$

More in general, let P be a set of LO rewrite rules over Σ and $\mathscr{V}$, and $\mathscr{M}, \mathscr{M}'$ two multisets of ground atomic formulas (two configurations). Furthermore, let H, G the (possibly empty) $\parr$-disjunctions of ground atomic formulas such that $\widehat{H} = \mathscr{M}'$ and $\widehat{G} = \mathscr{M}$. Then, the provability of the sequent $P, H \circ\!\!- \top \vdash G$ precisely characterizes the problem of *coverability* for the multiset (configuration) $\mathscr{M}'$, namely $P, H \circ\!\!- \top \vdash G$ is provable if and only if there exists a sequence of multiset rewriting steps defined over the theory P that, starting from $\mathscr{M}$, reaches a configuration $\mathscr{N}$ that *covers* $\mathscr{M}'$, i.e. such that $\mathscr{M}' \preccurlyeq \mathscr{N}$.

This is a straightforward consequence of the properties of clauses like $H \circ\!\!- \top$ (it succeeds only if a sub-multiset of the right-hand side of the current sequent matches $\widehat{H}$) and of the fact that, when working with LO rewrite rules, derivations have no branching. In other words the only way we can transform a partial derivation like the one in Figure 3 into a *proof* is to apply (once and only once since derivations form a single line) the clause with $\top$ (i.e. the target configuration is reached).

Coverability is strictly related to the verification problem of *safety properties* for concurrent systems (Abdulla *et al.*, 1996; Finkel and Schnoebelen, 2001). For instance, as shown in Bozzano *et al.* (2002), this property allows one to describe properties like *coverability* for a marking of a Petri Net. In Section 3.3, we will show how to exploit this property in the more general case of *first order* specifications.

In Section 8 we will discuss a possible characterization of *reachability* for two configurations using derivability in an extension of LO.

We conclude this section by discussing how *universal quantification* can be used in order to enrich the expressiveness of LO rewrite rules.

The Role of Universal Quantification In the *proofs as computations* interpretation of logic programs, universal quantification is a logical operator which provides a way to generate *new values*. From a logical perspective, this view of universal quantification is based on its proof-theoretical semantics in intuitionistic logic (Miller *et al.*, 1991). We will define first order rewrite rules with universal quantification taking inspiration

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

from Cervesato *et al.* (1999), where a similar logic fragment, called MSR, is defined. In Cervesato *et al.* (1999), MSR is used for the specification and analysis of *security protocols.*

Given the direct relationship between (first order) multiset rewriting and (first order) linear logic, it should be evident that multiset rewriting with universal quantification is the counterpart of LO with universal quantification. Having this idea in mind, we extend the notion of LO rewrite rule as follows.

Definition 3.11
We call *LO quantified rewrite rule* any LO formula having the following form

$$\forall(A_1 \parr \ldots \parr A_n \ \circ\!\!- \forall x_1, \ldots, x_n.(B_1 \parr \ldots \parr B_m))$$

where $A_1, \ldots, A_n$ and $B_1, \ldots, B_m$ are atomic formulas over Σ and $\mathscr{V}$.

The operational semantics of LO theories consisting of LO quantified rewrite rules should be clear by looking at the LO proof rule for universally quantified goal formulas: they are eliminated by introducing new constants. This operational behavior naturally corresponds to the extension of multiset rewriting with fresh name generation defined in Cervesato *et al.* (1999).

Remark 3.12
As mentioned at the beginning of this section, we remark that in (Cervesato *et al.*, 1999) the logic MSR is compared with a fragment of linear logic which turns out to be *dual* with respect to ours, and therefore *existential* quantification is used in place of universal quantification. Specifically, an MSR rule is defined as $A \multimap \exists x.\, B$, meaning that A evolves into B by creating a new name for x. In LO with universal quantification the same effect is obtained via the clause $A \circ\!\!- \forall x.\, B$. In fact, in the goal driven proof system of LO a computation step is obtained by resolution (i.e. reducing the conclusion of a clause to its premise).

Readers may refer to Cervesato (1994, 1995) and *et al.* (1999, 2000) for a more formal treatment of the relationship between multiset rewriting and LO.

3.2 Specification of concurrent systems

The connection with multiset rewriting allows us to think about LO as a specification language for concurrent systems. We will illustrate this idea with the help of the following example. We consider here a distributed *test-and-lock* protocol for a net with multiple resources, each of which is controlled by a monitor.

The protocol is as follows. A set of *resources*, distinguished by means of *resource identifiers*, and an arbitrary set of processes are given. Processes can non-deterministically request access to any resource. Access to a given resource must be exclusive (only one process at a time). Mutual exclusion is enforced by providing each resource with a *semaphore.*

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Given a propositional symbol *init*, we can encode the initial states of the system as follows:

1. $init \circ\!\!- init \parr think$
2. $init \circ\!\!- init \parr m(x, unlocked)$
3. $init \circ\!\!- \bot$

The atom *think* represents a thinking (idle) process, while the first order atom $m(x, s)$ represents a *monitor* for the resource with identifier x and associated semaphore s. The semaphore s can assume one of the two values *locked* or *unlocked*. Clause 1 and clause 2 can modify the initial state by adding, respectively, an arbitrary number of thinking processes and an arbitrary number of resources (with an initially unlocked semaphore). Finally, using clause 3 the atom *init* can be removed after the initialization phase.

The core of the protocol works as follows:

4. $think \circ\!\!- wait(x)$
5. $wait(x) \circ\!\!- think$
6. $wait(x) \parr m(x, unlocked) \circ\!\!- use(x) \parr m(x, locked)$
7. $use(x) \parr m(x, locked) \circ\!\!- think \parr m(x, unlocked)$

Using clause 4, a process can non-deterministically request access to any resource with identifier x, moving to a waiting state represented by the atom $wait(x)$. Clause 5 allows a process to go back to thinking from a waiting state. By clause 6, a waiting process can synchronize with the relevant monitor and is granted access provided the corresponding semaphore is unlocked. As a result, the semaphore is locked. The atom $use(x)$ represents a process which is currently using the resource with identifier x. Clause 7 allows a process to release a resource and go back to thinking, unlocking the corresponding semaphore.

Remark 3.13
In the previous specification we have intentionally introduced a flaw which we will disclose later (see Section 7). Uncovering of this flaw will allow us to explain and better motivate the use of the universal quantifier for the generation of new names.

3.3 Linear logic and model checking

One of the properties we would like to establish for the specification given in the previous example is that it ensures mutual exclusion for any resource used in the system. One of the difficulties for proving this kind of properties is that the specification taken into consideration has an infinite number of possible configurations (all possible rewritings of the goal *init*).

In this paper we shall define techniques that can be used to attack this kind of verification problems by exploiting an interesting connection between verification and bottom-up evaluation of LO programs.

Let us consider again the protocol specification given in Example 3.2. The mutual exclusion property can be formulated as the following property over reachable

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Infinite State Concurrent Systems	**Linear Logic Specification**
transition system	LO program and proof system
transition	rule instance
current state	goal formula
initial state	initial goal
upward-closed set of states	LO clause with $\top$
forward reachability	top-down provability
backward reachability	bottom-up provability

Fig. 4. Reachability versus provability.

configurations. Let S be the set of multiset of atomic formulas (i.e. a configuration) reachable in any derivation from the goal *init*. The protocol ensures mutual exclusion for resource x if and only if for any $\mathscr{A} \in S$, i.e. any reachable configuration, $\{use(x), use(x)\}$ is not a sub-multiset of $\mathscr{A}$. In other words, all goal formulas containing two occurrences of the formula $use(x)$ represent possible violations of mutual exclusion for resource x.

Following from the previous observation, a possible way of proving mutual exclusion for our sample protocol is to show that no configurations having the form $\{use(x), use(x), \ldots\}$ can be reached starting from the initial states. This verification methodology can be made effective using a backward exploration of a protocol specification as described in Abdulla *et al.* (1996). Specifically, the idea is to saturate the set of predecessor configurations (i.e. compute all possible predecessor configurations of the potential violations) and then check that no initial state occurs in the resulting set.

This verification strategy can be reformulated in a natural way in our fragment of linear logic. First of all, LO formulas with the $\top$ constant can be used to finitely represent all possible violations as follows:

$$U \stackrel{\text{def}}{=} \forall x.\ use(x) \parr use(x) \multimap \top$$

Backward reachability amounts then to compute all possible *logical consequences* of the LO specification of the protocol and of the formula U. In logic programming this strategy is called *bottom-up provability*. If the goal *init* is in the resulting set, then there exists an execution (derivation) terminated by an instance of the axiom $\top$ that leads from *init* to a multiset of the form $use(x), use(x), \mathscr{A}$ for some x and some multiset of atomic formulas $\mathscr{A}$. Thus, the use of clauses with $\top$ to represent violations (and admissibility of weakening) allows us to reason independently of the number of processes in the initial states. Following Abdulla *et al.* (1996), formulas like $\forall x.\, use(x) \parr use(x) \multimap \top$ can be viewed as a symbolic representation of *upward-closed* sets of configurations.

On the basis of these observations, the relationship between reachability and derivability sketched in the previous sections can be extended as shown in Figure 4.

To exploit this connection and extend the backward reachability strategy in the rest of the paper we will define a *bottom-up semantics* for first order LO programs. We will define our semantics via a fixpoint operator similar to the T_P operator used for logic programs. The fixpoint semantics will give us an effective way to evaluate

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

bottom-up an LO program, and thus solve verification problems for infinite-state concurrent systems as the one described in this section.

4 A Bottom-up semantics for $LO_\forall$

The proof-theoretical semantics for $LO_\forall$ corresponds to the *top-down* operational semantics based on resolution for traditional logic programming languages like Prolog. 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 $LO_\forall$ programs. More precisely, we will define an *effective* and *goal-independent* procedure to compute all goal formulas which are provable from a given program P. This semantics extends the one described in Bozzano *et al.* (2002), which was limited to propositional LO programs. In the following, given an $LO_\forall$ program P, we denote by Σ_P the signature comprising the set of constant, function, and predicate symbols in P.

4.1 Non-ground semantics for $LO_\forall$

Before discussing the bottom-up semantics, we lift the definition of operational semantics to $LO_\forall$ programs. Following Bozzano *et al.* (2002), we would like to define the operational semantics of a program P as the set of multisets of *atoms* which are provable from P. This could be done by considering the *ground instances* of $LO_\forall$ program clauses (see Definition 3.5). However, in presence of universal quantification in goals, this solution is not completely satisfactory. Consider, in fact, the following example. Take a signature with a predicate symbol p and two constants a and b, and consider the $LO_\forall$ program consisting of the axiom $\forall x.p(x) \circ\!\!- \top$ and the program consisting of the two clauses $p(a) \circ\!\!- \top$ and $p(b) \circ\!\!- \top$. The two programs would have the same *ground* semantics (i.e. consisting of the two singleton multisets $\{p(a)\}$ and $\{p(b)\}$). However, the $LO_\forall$ goal $\forall x.p(x)$ succeeds only in the first program, as the reader can verify. In order to distinguish the two programs, we need to consider the *non ground* semantics. In particular, our aim in this section will be to extend the so-called *C-semantics* of Falaschi *et al.* (1993) to first order LO.

First, we give the following definition.

Definition 4.1 (*Clause Variants*)
Given an $LO_\forall$ program P, the set of variants of clauses in P, denoted $Vrn(P)$, is defined as follows:

$$Vrn(P) \stackrel{\text{def}}{=} \{(H \circ\!\!- G)\,\theta \mid \forall\,(H \circ\!\!- G) \in P \text{ and } \theta \text{ is a renaming of the variables in } FV(H \circ\!\!- G) \text{ with new variables}\}.$$

Now, we need to reformulate the proof-theoretical semantics of Section 3 (see Figure 1). According to the C-semantics of Falaschi *et al.* (1993), our goal is to define the set of *non ground* goals which are provable from a given program P with an *empty answer substitution*. Slightly departing from Falaschi *et al.* (1993), we modify the proof system of Figure 1 as follows. Sequents are defined now over non ground goals. The backchaining rule of Figure 1 is replaced by the new rule shown

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$$\frac{P \vdash_\Sigma G\theta, \mathscr{A}}{P \vdash_\Sigma \widehat{H}\theta, \mathscr{A}} \quad bc \quad (H \circ\!\!- G) \in Vrn(P)$$

Fig. 5. Backchaining rule working over non ground goals.

in Figure 5 (where, as usual, $\mathscr{A}$ denotes a multiset of atomic formulas). The right-introduction rules and the axioms are as in Figure 1. This proof system is based on the idea of considering a first order program as the (generally *infinite*) collection of (*non ground*) instances of its clauses. By *instance* of a clause $H \circ\!\!- G$, we mean a clause $H\theta \circ\!\!- G\theta$, where θ is *any* substitution. The reader can see that, with this intuition, the set of goals provable from the system modified with the backchaining rule shown in Figure 5 corresponds to the set of non ground goals which are provable with an empty answer substitution according to Falaschi *et al.* (1993). This formulation of the proof system is the proof-theoretical counterpart of the bottom-up semantics we will define in the following.

All formulas (and also substitutions) on the right-hand side of the sequents in the proof system obtained from Figure 1 by replacing the backchaining rule with the rule of Figure 5 are implicitly assumed to range over the set of *non ground* terms over Σ. Every time rule $\forall_r$ is fired, a new constant c is added to the current signature, and the resulting goal is proved in the new signature (see Remark 3.7). Rule *bc* denotes a backchaining (resolution) step, where θ indicates *any* substitution. For our purposes, we can assume $Dom(\theta) \subseteq FV(H) \cup FV(G)$ (we remind that $FV(F)$ denotes the *free* variables of F). Note that $H \circ\!\!- G$ is assumed to be a variant, therefore it has no variables in common with $\mathscr{A}$. According to the usual concept of *uniformity*, *bc* can be executed only if the right-hand side of the current sequent consists of atomic formulas. Rules $\top_r$, $\parr_r$, $\&_r$ and $\bot_r$ are the same as in propositional LO. A sequent is provable if all branches of its proof tree terminate with instances of the $\top_r$ axiom.

Clearly, the proof system obtained by considering the rule of Figure 5 is not *effective*, however it will be sufficient for our purposes. An effective way to compute the set of goals which are provable from the above proof system will be discussed in Section 5.

We give the following definition, where $\vdash_\Sigma$ denotes the provability relation defined by the proof system of Figure 1 in which the backchaining rule has been replaced by the rule of Figure 5.

Definition 4.2 (*Operational Semantics*)
Given an $LO_\forall$ program P, its operational semantics, denoted $O(P)$, is given by

$$O(P) \stackrel{\text{def}}{=} \{\mathscr{A} \mid \mathscr{A} \text{ is a multiset of (non ground) atoms in } A^{\forall}_{\Sigma_P} \text{ and } P \vdash_{\Sigma_P} \mathscr{A}\}.$$

Intuitively, the set $O(P)$ is closed by *instantiation*, i.e. $\mathscr{A}\theta \in O(P)$ for any substitution θ, provided $\mathscr{A} \in O(P)$. Note that the operational semantics only include multisets of (non ground) *atoms*, therefore no connective (including the *universal quantifier*) can appear in the set $O(P)$. However, the intuition will be that the variables appearing in a multiset in $O(P)$ must be implicitly considered *universally quantified* (e.g. $\{p(x), q(x)\} \in O(P)$ implies that the goal $\forall x.(p(x) \parr q(x))$ is provable from P).

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Also note that the information on provable *facts* from a given program P is all we need to decide whether a general goal (possibly with nesting of connectives) is provable from P or not. In fact, according to $\mathrm{LO}_\forall$ proof-theoretical semantics, provability of a compound goal can always be reduced to provability of a finite set of atomic multisets.

4.2 Fixpoint semantics for $LO_\forall$

We will now discuss the *bottom-up* semantics. To deal with universal quantification (and therefore signature augmentation), we extend the definitions of Herbrand base and (concrete) interpretations given in Bozzano *et al.* (2002) as follows. Let Sig_P be the set of all possible extensions of the signature Σ_P associated to a program P with new constants. The definition of Herbrand base now depends explicitly on the signature, and interpretations can be thought of as infinite tuples, with one element for every signature $\Sigma \in Sig_P$. From here on the powerset of a given set D will be indicated as $\wp(D)$.

We give then the following definitions.

Definition 4.3 (*Herbrand Base*)
Given an $\mathrm{LO}_\forall$ program P and a signature $\Sigma \in Sig_P$, the Herbrand base of P over Σ, denoted $HB_\Sigma(P)$, is given by

$$HB_\Sigma(P) \stackrel{\text{def}}{=} \mathscr{MS}(A_\Sigma^{\not\forall}) = \{\mathscr{A} \mid \mathscr{A} \text{ is a multiset of (non ground) atoms in } A_\Sigma^{\not\forall}\}.$$

Definition 4.4 (*Interpretations*)
Given an $\mathrm{LO}_\forall$ program P, a (concrete) interpretation is a family of sets $\{I_\Sigma\}_{\Sigma \in Sig_P}$, where $I_\Sigma \in \wp(HB_\Sigma(P))$ for every $\Sigma \in Sig_P$.

In the following we often use the notation I for an interpretation to denote the family $\{I_\Sigma\}_{\Sigma \in Sig_P}$.

Interpretations form a complete lattice where inclusion and least upper bound are defined like (component-wise) set inclusion and union. In the following definition we therefore overload the symbols $\subseteq$ and $\cup$ for sets.

Definition 4.5 (*Interpretation Domain*)
Interpretations form a complete lattice $\langle \mathscr{D}, \subseteq \rangle$, where:

- $\mathscr{D} = \{I \mid I \text{ is an interpretation}\}$;
- $I \subseteq J$ if and only if $I_\Sigma \subseteq J_\Sigma$ for every $\Sigma \in Sig_P$;
- the least upper bound of I and J is $\{I_\Sigma \cup J_\Sigma\}_{\Sigma \in Sig_P}$;
- the bottom and top elements are $\emptyset = \{\emptyset_\Sigma\}_{\Sigma \in Sig_P}$ and $\{HB_\Sigma(P)\}_{\Sigma \in Sig_P}$, respectively.

Before introducing the definition of fixpoint operator, we need to define the notion of satisfiability of a context Δ (a multiset of goal formulas) in a given interpretation I. For this purpose, we introduce the judgment $I \models_\Sigma \Delta \blacktriangleright \mathscr{C}$, where I is an *input* interpretation, Δ is an *input* context, and $\mathscr{C}$ is an *output* fact (a multiset of atomic formulas). The judgment is also parametric with respect to a given signature Σ.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

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 $LO_\forall$ clause bodies. The satisfiability judgment is modeled according to the right-introduction rules of the connectives. In other words, the computation performed by the satisfiability judgment corresponds to *top-down* steps inside our *bottom-up* semantics. Intuitively, the parameter $\mathscr{C}$ must be thought of as an *output* fact such that $\mathscr{C}+\Delta$ is valid in I. The notion of output fact will simplify the presentation of the algorithmic version of the judgment which we will present in Section 5. The notion of *satisfiability* is modeled according to the right-introduction (decomposition) rules of the proof system, as follows (we remind that '+' denotes multiset union).

Definition 4.6 (*Satisfiability Judgment*)
Let P be an $LO_\forall$ program, $\Sigma \in Sig_P$, and $I = \{I_\Sigma\}_{\Sigma \in Sig_P}$ an interpretation. The satisfiability judgment $\models_\Sigma$ is defined as follows:

$I \models_\Sigma \top, \Delta \blacktriangleright \mathscr{C}$ for any fact $\mathscr{C}$ in A^{γ}_{Σ};

$I \models_\Sigma \mathscr{A} \blacktriangleright \mathscr{C}$ if $\mathscr{A} + \mathscr{C} \in I_\Sigma$;

$I \models_\Sigma \forall x.G, \Delta \blacktriangleright \mathscr{C}$ if $I \models_{\Sigma,c} G[c/x], \Delta \blacktriangleright \mathscr{C}$, with $c \notin \Sigma$ (see remark 4.7);

$I \models_\Sigma G_1 \,\&\, G_2, \Delta \blacktriangleright \mathscr{C}$ if $I \models_\Sigma G_1, \Delta \blacktriangleright \mathscr{C}$ and $I \models_\Sigma G_2, \Delta \blacktriangleright \mathscr{C}$;

$I \models_\Sigma G_1 \parr G_2, \Delta \blacktriangleright \mathscr{C}$ if $I \models_\Sigma G_1, G_2, \Delta \blacktriangleright \mathscr{C}$;

$I \models_\Sigma \bot, \Delta \blacktriangleright \mathscr{C}$ if $I \models_\Sigma \Delta \blacktriangleright \mathscr{C}$.

Remark 4.7
When using the notation $I \models_\Sigma \Delta \blacktriangleright \mathscr{C}$ we *always* make the *implicit* assumption that Δ is a context defined over Σ (i.e. term constructors in Δ must belong to Σ). As a result, also the output fact $\mathscr{C}$ must be defined over Σ. This assumption, which is the counterpart (see Remark 3.7) of an analogous assumption for proof systems like the one in Figure 1, i.e. with explicit signature notation, will *always* and *tacitly* hold in the following. For example, note that in the $\forall$-case of the $\models_\Sigma$ definition below, the newly introduced constant c *cannot be exported* through the output fact $\mathscr{C}$. This is crucial to capture the operational semantics of the universal quantifier.

The satisfiability judgment $\models_\Sigma$ satisfies the following properties.

Lemma 1
For every interpretation $I = \{I_\Sigma\}_{\Sigma \in Sig_P}$, context Δ, and fact $\mathscr{C}$,

$$I \models_\Sigma \Delta \blacktriangleright \mathscr{C} \text{ if and only if } I \models_\Sigma \Delta, \mathscr{C} \blacktriangleright \epsilon.$$

Proof
See Bozzano *et al.* (2003). □

Lemma 2
For any interpretations $I_1 = \{(I_1)_\Sigma\}_{\Sigma \in Sig_P}, I_2 = \{(I_2)_\Sigma\}_{\Sigma \in Sig_P}, \ldots$, context Δ, and fact $\mathscr{C}$,

i. if $I_1 \subseteq I_2$ and $I_1 \models_\Sigma \Delta \blacktriangleright \mathscr{C}$ then $I_2 \models_\Sigma \Delta \blacktriangleright \mathscr{C}$;
ii. if $I_1 \subseteq I_2 \subseteq \ldots$ and $\bigcup_{i=1}^{\infty} I_i \models_\Sigma \Delta \blacktriangleright \mathscr{C}$ then there exists $k \in \mathbb{N}$ s.t. $I_k \models_\Sigma \Delta \blacktriangleright \mathscr{C}$.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Proof
See Bozzano *et al.* (2003). □

We are now ready to define the fixpoint operator T_P.

Definition 4.8 (*Fixpoint Operator* $\boldsymbol{T_P}$)
Given an $LO_\forall$ program P and an interpretation $I = \{I_\Sigma\}_{\Sigma \in Sig_P}$, the fixpoint operator T_P is defined as follows:

$$T_P(I) \stackrel{\text{def}}{=} \{(T_P(I))_\Sigma\}_{\Sigma \in Sig_P};$$

$$(T_P(I))_\Sigma \stackrel{\text{def}}{=} \{\widehat{H}\theta + \mathscr{C} \mid (H \circ\!\!- G) \in Vrn(P),\ \theta \text{ is any substitution, and } I \models_\Sigma G\theta \blacktriangleright \mathscr{C}\}.$$

Remark 4.9
In the previous definition, θ is implicitly assumed to be defined over Σ, i.e. θ can only map variables in $Dom(\theta)$ to terms in $T_\Sigma^\mathscr{V}$.

The following property holds.

Proposition 4.10 (*Monotonicity and Continuity*)
For every $LO_\forall$ program P, the fixpoint operator T_P is monotonic and continuous over the lattice $\langle \mathscr{D}, \subseteq \rangle$.

Proof
Monotonicity.
Immediate from the definition of T_P and item *i* of Lemma 2.

Continuity.
We prove that T_P is finitary, i.e. for any sequence of interpretations $I_1 \subseteq I_2 \subseteq \ldots$ we have that $T_P(\bigcup_{i=1}^\infty I_i) \subseteq \bigcup_{i=1}^\infty T_P(I_i)$, i.e. for every $\Sigma \in \Sigma_P$, $(T_P(\bigcup_{i=1}^\infty I_i))_\Sigma \subseteq (\bigcup_{i=1}^\infty T_P(I_i))_\Sigma$. Let $\mathscr{A} \in (T_P(\bigcup_{i=1}^\infty I_i))_\Sigma$. By definition of T_P, there exist a variant $H \circ\!\!- G$ of a clause in P, a substitution θ, and a fact $\mathscr{C}$ s.t. $\mathscr{A} = \widehat{H}\theta + \mathscr{C}$ and $\bigcup_{i=1}^\infty I_i \models_\Sigma G\theta \blacktriangleright \mathscr{C}$. By item *ii* of Lemma 2, we have that there exists $k \in \mathbb{N}$ s.t. $I_k \models_\Sigma G\theta \blacktriangleright \mathscr{C}$. Again by definition of T_P, we get $\mathscr{A} = \widehat{H}\theta + \mathscr{C} \in (T_P(I_k))_\Sigma \subseteq (\bigcup_{i=1}^\infty T_P(I_i))_\Sigma$. □

Monotonicity and continuity of the T_P operator imply, by Tarski's Theorem, that $lfp(T_P) = T_P{\uparrow}_\omega$. The fixpoint semantics of a program P is then defined as follows.

Definition 4.10 (*Fixpoint Semantics*)
Given an $LO_\forall$ program P, its fixpoint semantics, denoted $F(P)$, is defined as follows:

$$F(P) \stackrel{\text{def}}{=} (lfp(T_P))_{\Sigma_P} = (T_P{\uparrow}_\omega(\{\emptyset_\Sigma\}_{\Sigma \in Sig_P}))_{\Sigma_P}.$$

We conclude this section by proving the following fundamental result, which states that the fixpoint semantics is sound and complete with respect to the operational semantics (see Definition 4.2).

Theorem 1 (*Soundness and Completeness*)
For every $LO_\forall$ program P, $F(P) = O(P)$.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Proof

$F(P) \subseteq O(P)$.

We prove that for every $k \in \mathbb{N}$, for every signature $\Sigma \in Sig_P$, and for every context Δ, $T_P\uparrow_k \models_\Sigma \Delta \blacktriangleright \epsilon$ implies $P \vdash_\Sigma \Delta$. The proof is by lexicographic induction on (k, h), where h is the length of the derivation of $T_P\uparrow_k \models_\Sigma \Delta \blacktriangleright \epsilon$.

- If $\Delta = \top, \Delta'$, obvious;
- if $\Delta = \mathscr{A}$ and $\mathscr{A} \in (T_P\uparrow_k)_\Sigma$, then there exist a variant $H \circ\!\!- G$ of a clause in P, a fact $\mathscr{C}$ and a substitution θ s.t. $\mathscr{A} = \widehat{H}\theta + \mathscr{C}$ and $T_P\uparrow_{k-1} \models_\Sigma G\theta \blacktriangleright \mathscr{C}$. By Lemma 1, this implies $T_P\uparrow_{k-1} \models_\Sigma G\theta, \mathscr{C} \blacktriangleright \epsilon$. Then by the inductive hypothesis we have $P \vdash_\Sigma G\theta, \mathscr{C}$, from which $P \vdash_\Sigma \widehat{H}\theta, \mathscr{C}$, i.e. $P \vdash_\Sigma \mathscr{A}$ follows by bc rule;
- if $\Delta = \forall x.G, \Delta'$ and $T_P\uparrow_k \models_{\Sigma,c} G[c/x], \Delta' \blacktriangleright \epsilon$, with $c \notin \Sigma$, then by the inductive hypothesis we have $P \vdash_{\Sigma,c} G[c/x], \Delta'$ from which $P \vdash_\Sigma \forall x.G, \Delta'$ follows by $\forall_r$ rule;
- if $\Delta = G_1 \,\&\, G_2, \Delta'$, $T_P\uparrow_k \models_\Sigma G_1, \Delta' \blacktriangleright \epsilon$, and $T_P\uparrow_k \models_\Sigma G_2, \Delta' \blacktriangleright \epsilon$, then by the inductive hypothesis we have $P \vdash_\Sigma G_1, \Delta'$ and $P \vdash_\Sigma G_2, \Delta'$, from which $P \vdash_\Sigma G_1 \,\&\, G_2, \Delta'$ follows by $\&_r$ rule;
- if $\Delta = G_1 \parr G_2, \Delta'$ and $T_P\uparrow_k \models_\Sigma G_1, G_2, \Delta' \blacktriangleright \epsilon$, then by the inductive hypothesis we have $P \vdash_\Sigma G_1, G_2, \Delta'$, from which $P \vdash_\Sigma G_1 \parr G_2, \Delta'$ follows by $\parr_r$ rule;
- if $\Delta = \bot, \Delta'$ and $T_P\uparrow_k \models_\Sigma \Delta' \blacktriangleright \epsilon$, then by the inductive hypothesis we have $P \vdash_\Sigma \Delta'$, from which $P \vdash_\Sigma \bot, \Delta'$ follows by $\bot_r$ rule.

$O(P) \subseteq F(P)$.

We prove that for every signature $\Sigma \in Sig_P$ and for every context Δ, if $P \vdash_\Sigma \Delta$ then there exists $k \in \mathbb{N}$ s.t. $T_P\uparrow_k \models_\Sigma \Delta \blacktriangleright \epsilon$. The proof is by induction on the derivation of $P \vdash_\Sigma \Delta$.

- If $\Delta = \top, \Delta'$, then for every $k \in \mathbb{N}$, $T_P\uparrow_k \models_\Sigma \Delta \blacktriangleright \epsilon$;
- if $\Delta = \widehat{H}\theta, \mathscr{A}$, with $H \circ\!\!- G$ a variant of a clause in P, θ substitution, and $P \vdash_\Sigma G\theta, \mathscr{A}$, then by the inductive hypothesis we have that there exists $k \in \mathbb{N}$ s.t. $T_P\uparrow_k \models_\Sigma G\theta, \mathscr{A} \blacktriangleright \epsilon$. Then, by Lemma 1, $T_P\uparrow_k \models_\Sigma G\theta \blacktriangleright \mathscr{A}$. By definition of T_P, $\widehat{H}\theta + \mathscr{A} \in (T_P\uparrow_{k+1})_\Sigma$, which implies $T_P\uparrow_{k+1} \models_\Sigma \widehat{H}\theta + \mathscr{A} \blacktriangleright \epsilon$;
- if $\Delta = \forall x.G, \Delta'$ and $P \vdash_{\Sigma,c} G[c/x], \Delta'$, with $c \notin \Sigma$, then by the inductive hypothesis we have that there exist $k \in \mathbb{N}$ s.t. $T_P\uparrow_k \models_{\Sigma,c} G[c/x], \Delta' \blacktriangleright \epsilon$, from which $T_P\uparrow_k \models_\Sigma \forall x.G, \Delta' \blacktriangleright \epsilon$ follows;
- if $\Delta = G_1 \,\&\, G_2, \Delta'$, $P \vdash_\Sigma G_1, \Delta'$ and $P \vdash_\Sigma G_2, \Delta'$, then by the inductive hypothesis we have that there exist $k_1, k_2 \in \mathbb{N}$ s.t. $T_P\uparrow_{k_1} \models_\Sigma G_1, \Delta' \blacktriangleright \epsilon$ and $T_P\uparrow_{k_2} \models_\Sigma G_2, \Delta' \blacktriangleright \epsilon$. By taking $k = max\{k_1, k_2\}$, by item i of Lemma 2 and monotonicity of T_P (Proposition 2) we get $T_P\uparrow_k \models_\Sigma G_1, \Delta' \blacktriangleright \epsilon$ and $T_P\uparrow_k \models_\Sigma G_2, \Delta' \blacktriangleright \epsilon$, from which $T_P\uparrow_k \models_\Sigma G_1 \,\&\, G_2, \Delta' \blacktriangleright \epsilon$ follows;
- if $\Delta = G_1 \parr G_2, \Delta'$ or $\Delta = \bot, \Delta'$, the conclusion follows by a straightforward application of the inductive hypothesis.

□

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Example 4.11
Let Σ be a signature including the constant symbols a and b, a function symbol f, and the predicate symbols p, q, r, let $\mathscr{V}$ be a denumerable set of variables and $x, y \in \mathscr{V}$, and let P be the following $\mathrm{LO}_\forall$ program:

1. $r(f(b)) \parr p(a) \circ\!\!- \top$
2. $p(x) \circ\!\!- \top$
3. $q(y) \circ\!\!- (\forall x.p(x)) \,\&\, r(y)$

Let $I_0 = \{\emptyset_\Sigma\}_{\Sigma \in Sig_P}$, and let us compute $I_1 = T_P(I_0)$. Using clauses 1 and 2, we get that (see Definitions 4.6 and 4.8) $(I_1)_\Sigma$ contains the multisets of atoms of the form $\{r(f(b)), p(a)\} + \mathscr{A}$, and $\{p(t)\} + \mathscr{A}$, where $\mathscr{A}$ is any multiset of (possibly non-ground) atoms in $A^{\mathscr{V}}_\Sigma$, while t is any (possibly non ground) term in $T^{\mathscr{V}}_\Sigma$. Similarly $(I_1)_{\Sigma'}$, for a generic signature Σ' such that $\Sigma \subseteq \Sigma'$, contains all multisets of the above form where $\mathscr{A}$ and t are taken from, respectively, $A^{\mathscr{V}}_{\Sigma'}$ and $T^{\mathscr{V}}_{\Sigma'}$. For instance, let c be a new constant not appearing in Σ. The set $(I_1)_{\Sigma'}$ will contain, e.g. the multisets $\{p(c)\}$, $\{p(f(c)), q(b)\}$, and so on.

Now, consider the substitution $\theta = [y \mapsto f(b)]$ and the following corresponding instance of clause 3: $q(f(b)) \circ\!\!- (\forall x.p(x)) \,\&\, r(f(b))$. Assume we want to compute an output fact $\mathscr{C}$ for the judgment

$$I_1 \models_\Sigma (\forall x.p(x)) \,\&\, r(f(b)) \blacktriangleright \mathscr{C}.$$

By definition of $\models$, we have to compute $I_1 \models_\Sigma (\forall x.p(x)) \blacktriangleright \mathscr{C}$ and $I_1 \models_\Sigma r(f(b)) \blacktriangleright \mathscr{C}$. For the latter judgment we have that, e.g. $I_1 \models_\Sigma r(f(b)) \blacktriangleright p(a)$. For the first judgment, by definition of $\models$, we must compute $I_1 \models_{\Sigma,c} p(c) \blacktriangleright \mathscr{C}$, where c is a new constant not in Σ. As $\{p(c)\}$ is contained in $(I_1)_{\Sigma,c}$, we can get that $I_1 \models_{\Sigma,c} p(c) \blacktriangleright \epsilon$. We can also get $I_1 \models_{\Sigma,c} p(c) \blacktriangleright p(a)$ (in fact $\{p(c), p(a)\}$ is also contained in $(I_1)_{\Sigma,c}$. By applying the $\&$-rule for $\models$, we get that $I_1 \models_\Sigma (\forall x.p(x)) \,\&\, r(f(b)) \blacktriangleright p(a)$. Therefore, by applying clause 3 we get that, e.g. the multiset $\{q(b), p(a)\}$ is in $(I_2)_\Sigma = (T_P(I_1))_\Sigma$. □

5 An effective semantics for $\mathrm{LO}_\forall$

The fixpoint 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. This is a result of both the definition of the satisfiability judgment (whose clause for $\top$ is clearly not effective) and the definition of interpretations as infinite tuples. In order to solve these problems, we first define the (abstract) Herbrand base and (abstract) interpretations as follows.

Definition 5.1 (*Abstract Herbrand Base*)
Given an $\mathrm{LO}_\forall$ program P, the Herbrand base of P, denoted $HB(P)$, is given by

$$HB(P) \stackrel{\text{def}}{=} HB_{\Sigma_P}(P).$$

Definition 5.2 (*Abstract Interpretations*)
Given an $\mathrm{LO}_\forall$ program P, an interpretation I is any subset of $HB(P)$, i.e. $I \in \wp(HB(P))$.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

To define the abstract domain of interpretations, we need the following definitions.

Definition 5.3 (*Instance Operator*)
Given an interpretation I and a signature $\Sigma \in Sig_P$, we define the operator $Inst_\Sigma$ as follows:

$$Inst_\Sigma(I) = \{\mathscr{A}\theta \mid \mathscr{A} \in I,\ \theta \text{ substitution over } \Sigma\}.$$

Definition 5.4 (*Upward-closure Operator*)
Given an interpretation I and a signature $\Sigma \in Sig_P$, we define the operator Up_Σ as follows:

$$Up_\Sigma(I) = \{\mathscr{A} + \mathscr{C} \mid \mathscr{A} \in I,\ \mathscr{C} \text{ fact over } \Sigma\}.$$

Remark 5.5
Note that, as usual, in the previous definitions we assume the substitution θ and the fact $\mathscr{C}$ to be defined over the signature Σ.

The following definition provides the connection between the (abstract) interpretations defined in Definition 5.2 and the (concrete) interpretations of Definition 4.4. The idea behind the definition is that an interpretation implicitly *denotes* the set of elements which can be obtained by either *instantiating* or *closing upwards* elements in the interpretation itself (where the concepts of instantiation and upward-closure are made precise by the above definitions). The operation of instantiation is related to the notion of C-semantics (Falaschi *et al.*, 1993) (see Definition 4.2), while the operation of upward-closure is justified by Proposition 1. Note that the operations of instantiation and upward-closure are performed for every possible signature $\Sigma \in Sig_P$.

Definition 5.6 (*Denotation of an Interpretation*)
Given an (abstract) interpretation I, its denotation $[\![I]\!]$ is the (concrete) interpretation $\{[\![I]\!]_\Sigma\}_{\Sigma \in Sig_P}$ defined as follows:

$$[\![I]\!]_\Sigma \stackrel{\text{def}}{=} Inst_\Sigma(Up_\Sigma(I)) \quad \text{(or, equivalently, } [\![I]\!]_\Sigma \stackrel{\text{def}}{=} Up_\Sigma(Inst_\Sigma(I))\text{)}.$$

Two interpretations I and J are said to be equivalent, written $I \simeq J$, if and only if $[\![I]\!] = [\![J]\!]$.

The equivalence of the two different equations in Definition 5.6 is stated in the following proposition.

Proposition 3
For every interpretation I, and signature $\Sigma \in Sig_P$,

$$Inst_\Sigma(Up_\Sigma(I)) = Up_\Sigma(Inst_\Sigma(I)).$$

Proof
Let $(\mathscr{A} + \mathscr{C})\theta \in Inst_\Sigma(Up_\Sigma(I))$, with $\mathscr{A} \in I$. Then $(\mathscr{A} + \mathscr{C})\theta = (\mathscr{A}\theta) + \mathscr{C}\theta \in Up_\Sigma(Inst_\Sigma(I))$. Conversely, let $\mathscr{A}\theta + \mathscr{C} \in Up_\Sigma(Inst_\Sigma(I))$, with $\mathscr{A} \in I$. Let $\mathscr{B}$ be a variant of $\mathscr{C}$ with new variables (not appearing in $\mathscr{A}$, θ, and $\mathscr{C}$) and θ' be the substitution with domain $Dom(\theta) \cup FV(\mathscr{B})$ and s.t. $\theta'_{|Dom(\theta)} = \theta$ and θ' maps $\mathscr{B}$ to $\mathscr{C}$. Then $\mathscr{A}\theta + \mathscr{C} = \mathscr{A}\theta' + \mathscr{B}\theta' = (\mathscr{A} + \mathscr{B})\theta' \in Inst_\Sigma(Up_\Sigma(I))$. □

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

We are now ready to define the symbolic interpretation domain. In the following we will use the word *abstract* to stress the connection between our symbolic semantics and the theory of *abstract interpretation*. Our abstraction does not loose precision but it allows us to finitely represent infinite collections of formulas. As previously mentioned, the idea is that of considering interpretations as implicitly defining the sets of elements contained in their denotations. Therefore, differently from Definition 4.5, now we need to check containment between *denotations*. Furthermore, as we do not need to distinguish between interpretations having the same denotation, we simply identify them using equivalence classes with respect to the corresponding equivalence relation $\simeq$.

Definition 5.7 (*Abstract Interpretation Domain*)
Abstract interpretations form a complete lattice $\langle \mathscr{I}, \sqsubseteq \rangle$, where

- $\mathscr{I} = \{[I]_\simeq \mid I \text{ is an interpretation}\}$;
- $[I]_\simeq \sqsubseteq [J]_\simeq$ if and only if $[\![I]\!] \subseteq [\![J]\!]$;
- the least upper bound of $[I]_\simeq$ and $[J]_\simeq$, written $[I]_\simeq \bigsqcup [J]_\simeq$, is $[I \cup J]_\simeq$;
- the bottom and top elements are $[\emptyset]_\simeq$ and $[\epsilon]_\simeq$, respectively.

The following proposition provides an *effective* and equivalent condition for testing the $\sqsubseteq$ relation (which we call *entailment* relation) over interpretations. We will need this result later on.

Proposition 4 (*Entailment between Interpretations*)
Given two interpretations I and J, $[\![I]\!] \subseteq [\![J]\!]$ if and only if for every $\mathscr{A} \in I$, there exist $\mathscr{B} \in J$, a substitution θ, and a fact $\mathscr{C}$ (defined over Σ_P) s.t. $\mathscr{A} = \mathscr{B}\theta + \mathscr{C}$.

Proof
If part. We prove that for every $\Sigma \in Sig_P$, $[\![I]\!]_\Sigma \subseteq [\![J]\!]_\Sigma$. Let $\mathscr{A}' = \mathscr{A}\theta' + \mathscr{C}' \in Up_\Sigma(Inst_\Sigma(I)) = [\![I]\!]_\Sigma$, with $\mathscr{A} \in I$ and θ', $\mathscr{C}'$ defined over Σ. By hypothesis, there exist $\mathscr{B} \in J$, a substitution θ, and a fact $\mathscr{C}$ (defined over Σ_P) s.t. $\mathscr{A} = \mathscr{B}\theta + \mathscr{C}$. Therefore, $\mathscr{A}' = \mathscr{A}\theta' + \mathscr{C}' = (\mathscr{B}\theta + \mathscr{C})\theta' + \mathscr{C}' = \mathscr{B}\theta\theta' + (\mathscr{C}\theta' + \mathscr{C}') \in Up_\Sigma(Inst_\Sigma(J)) = [\![J]\!]_\Sigma$ (note that $\theta\theta'$ and $\mathscr{C}\theta' + \mathscr{C}'$ are both defined over Σ because $\Sigma_P \subseteq \Sigma$).

Only if part. Let $\mathscr{A} \in I$, then $\mathscr{A} \in [\![I]\!]_{\Sigma_P}$ (note that $\mathscr{A}$ is defined over Σ_P by definition of interpretation). Then, by the hypothesis we have that $\mathscr{A} \in [\![J]\!]_{\Sigma_P} = Up_{\Sigma_P}(Inst_{\Sigma_P}(J))$, i.e. there exist $\mathscr{B} \in J$, a substitution θ, and a fact $\mathscr{C}$ (defined over Σ_P) s.t. $\mathscr{A} = \mathscr{B}\theta + \mathscr{C}$. □

We now define the abstract satisfiability judgment $I \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$, where I is an *input* interpretation, Δ is an *input* context, $\mathscr{C}$ is an *output* fact, and θ is an *output* substitution.

Remark 5.8
As usual, the notation $I \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ requires that Δ, $\mathscr{C}$, and θ are defined over the signature Σ. As a consequence, the newly introduced constant c in the $\forall$-case of the $\Vdash_\Sigma$ definition below *cannot be exported* through the output parameters $\mathscr{C}$ or θ.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

The judgment $\Vdash_\Sigma$ can be thought of as an abstract version of the judgment $\models_\Sigma$ (compare Definition 4.6). We now need one more parameter, namely an *output* substitution. The idea behind the definition is that the output fact $\mathscr{C}$ and the output substitution θ are *minimal* (in a sense to be clarified) so that they can be computed effectively given a program P, an interpretation I, and a signature Σ. The output substitution θ is needed in order to deal with clause instantiation, and its minimality is ensured by using most general unifiers in the definition. As the reader can note, the sources of non-effectiveness which are present in Definition 4.6 (e.g. in the rule for $\top$)) are removed in Definition 5.9 below. We recall that the notation $\theta_1 \uparrow \theta_2$ denotes the least upper bound of substitutions (see Appendix A).

Definition 5.9 (*Abstract Satisfiability Judgment*)
Let P be an $\mathrm{LO}_\forall$ program, I an interpretation, and $\Sigma \in Sig_P$. The abstract satisfiability judgment $\Vdash_\Sigma$ is defined as follows:

$I \Vdash_\Sigma \top, \Delta \blacktriangleright \epsilon \blacktriangleright nil$;

$I \Vdash_\Sigma \mathscr{A} \blacktriangleright \mathscr{C} \blacktriangleright \theta$ if there exist $\mathscr{B} \in I$ (variant), $\mathscr{B}' \preccurlyeq \mathscr{B}$, $\mathscr{A}' \preccurlyeq \mathscr{A}$, $|\mathscr{B}'| = |\mathscr{A}'|$, $\mathscr{C} = \mathscr{B} \setminus \mathscr{B}'$, and $\theta = mgu(\mathscr{B}', \mathscr{A}')_{|FV(\mathscr{A},\mathscr{C})}$;

$I \Vdash_\Sigma \forall x.G, \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ if $I \Vdash_{\Sigma,c} G[c/x], \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$, with $c \notin \Sigma$ (see Remark 5.8);

$I \Vdash_\Sigma G_1 \,\&\, G_2, \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ if $I \Vdash_\Sigma G_1, \Delta \blacktriangleright \mathscr{C}_1 \blacktriangleright \theta_1$, $I \Vdash_\Sigma G_2, \Delta \blacktriangleright \mathscr{C}_2 \blacktriangleright \theta_2$, $\mathscr{D}_1 \preccurlyeq \mathscr{C}_1$, $\mathscr{D}_2 \preccurlyeq \mathscr{C}_2$, $|\mathscr{D}_1| = |\mathscr{D}_2|$, $\theta_3 = mgu(\mathscr{D}_1, \mathscr{D}_2)$, $\mathscr{C} = \mathscr{C}_1 + (\mathscr{C}_2 \setminus \mathscr{D}_2)$, and $\theta = (\theta_1 \uparrow \theta_2 \uparrow \theta_3)_{|FV(G_1,G_2,\Delta,\mathscr{C})}$;

$I \Vdash_\Sigma G_1 \parr G_2, \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ if $I \Vdash_\Sigma G_1, G_2, \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$;

$I \Vdash_\Sigma \bot, \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ if $I \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$.

We recall that two multisets in general may have more than one (not necessarily equivalent) most general unifier and that using the notation $mgu(\mathscr{B}', \mathscr{A}')$ we mean any unifier which is *non-deterministically* picked from the set of most general unifiers of $\mathscr{B}'$ and $\mathscr{A}'$ (see Appendix A).

Example 5.10
Let us consider a signature with a function symbol f and predicate symbols p, q, r, s. Let $\mathscr{V}$ be a denumerable set of variables, and $u, v, w, \ldots \in \mathscr{V}$. Let I be the interpretation consisting of the two multisets $\{p(x), q(x)\}$ and $\{r(y), p(f(y))\}$ (for simplicity, hereafter we omit brackets in multiset notation), and P the program

1. $r(w) \multimap q(f(w))$
2. $s(z) \multimap \forall x.p(f(x))$
3. $\bot \multimap q(u) \,\&\, r(v)$

Let us consider (a renaming of) the body of the first clause, $q(f(w'))$, and (a renaming of) the first element in I, $p(x'), q(x')$. Using the second case for the $\Vdash_{\Sigma_P}$ judgment, with $\mathscr{A} = \mathscr{A}' = q(f(w'))$, $\mathscr{B} = p(x'), q(x')$, $\mathscr{B}' = q(x')$, we get

$$I \Vdash_{\Sigma_P} q(f(w')) \blacktriangleright p(x') \blacktriangleright [x' \mapsto f(w')].$$

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Let us consider now (a renaming of) the body of the second case, $\forall x.p(f(x))$, and another renaming of the first element, $p(x''), q(x'')$. From the $\forall$-case of the definition of $\Vdash_{\Sigma_P}$, $I \Vdash_{\Sigma_P} \forall x.p(f(x)) \blacktriangleright \mathscr{C} \blacktriangleright \theta$ if $I \Vdash_{\Sigma_P,c} p(f(c)) \blacktriangleright \mathscr{C} \blacktriangleright \theta$, with $c \notin \Sigma_P$. Now, we can apply the second case for $\Vdash_{\Sigma_P,c}$. Unfortunately, we can't choose $\mathscr{A}'$ to be $p(f(c))$ and $\mathscr{B}'$ to be $p(x'')$. In fact, by unifying $p(f(c))$ with $p(x'')$, we should get the substitution $\theta = [x'' \mapsto f(c)]$ and the output fact $q(x'')$ (note that x'' is a free variable in the output fact) and this is not allowed because the substitution θ must be defined on Σ_P, in order for $I \Vdash_{\Sigma_P} \forall x.p(f(x)) \blacktriangleright \mathscr{C} \blacktriangleright \theta$ to be meaningful. It turns out that the only way to use the second clause for $\Vdash_{\Sigma_P,c}$ is to choose $\mathscr{A}' = \mathscr{B}' = \epsilon$, which is useless in the fixpoint computation (see Example 5.13). Finally, let us consider (a renaming of) the body of the third clause, $\bot \circ\!\!- q(u') \,\&\, r(v')$. According to the $\&$-rule for the $\Vdash_{\Sigma_P}$ judgment, we must first compute $\mathscr{C}_1, \mathscr{C}_2, \theta_1$ and θ_2 such that $I \Vdash_{\Sigma_P} q(u') \blacktriangleright \mathscr{C}_1 \blacktriangleright \theta_1$ and $I \Vdash_{\Sigma_P} r(v') \blacktriangleright \mathscr{C}_2 \blacktriangleright \theta_2$. To this aim, take two variants of the multisets in I, $p(x'''), q(x''')$ and $r(y'), p(f(y'))$. Proceeding as above, we get that

$$I \Vdash_{\Sigma_P} q(u') \blacktriangleright p(x''') \blacktriangleright [u' \mapsto x'''] \quad \text{and} \quad I \Vdash_{\Sigma_P} r(v') \blacktriangleright p(f(y')) \blacktriangleright [v' \mapsto y'].$$

Now, we can apply the $\&$-rule for the $\Vdash_{\Sigma_P}$ judgment, with $\mathscr{D}_1 = p(x''')$, $\mathscr{D}_2 = p(f(y'))$, and $\theta_3 = [x''' \mapsto f(y')]$. We have that $\theta_1 \uparrow \theta_2 \uparrow \theta_3 = [u' \mapsto f(y'), v' \mapsto y', x''' \mapsto f(y')]$. Therefore, we get that

$$I \Vdash_{\Sigma_P} q(u') \,\&\, r(v') \blacktriangleright p(x''') \blacktriangleright [u' \mapsto f(y'), v' \mapsto y', x''' \mapsto f(y')].$$

□

The following lemma states a simple property of the substitution domain, which we will need in the following.

Lemma 3
For every interpretation I, context Δ, fact $\mathscr{C}$, and substitution θ, if $I \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ then $Dom(\theta) \subseteq FV(\Delta) \cup FV(\mathscr{C})$.

Proof
Immediate by induction on the definition of $\Vdash_\Sigma$. □

The connection between the satisfiability judgments $\models_\Sigma$ and $\Vdash_\Sigma$ is clarified by the following lemma (in the following we denote by $\succcurlyeq$ the converse of the sub-multiset relation, i.e. $\mathscr{A} \succcurlyeq \mathscr{B}$ if and only if $\mathscr{B} \preccurlyeq \mathscr{A}$).

Lemma 4
For every interpretation I, context Δ, fact $\mathscr{C}$, and substitution θ,

i. if $I \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ then $[\![I]\!] \models_\Sigma \Delta\theta\theta' \blacktriangleright \mathscr{C}'\theta'$ for every substitution θ' and fact $\mathscr{C}' \succcurlyeq \mathscr{C}\theta$;
ii. if $[\![I]\!] \models_\Sigma \Delta\theta \blacktriangleright \mathscr{C}$ then there exist a fact $\mathscr{C}'$, and substitutions θ' and σ s.t. $I \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C}' \blacktriangleright \theta'$, $\theta_{|FV(\Delta)} = (\theta' \circ \sigma)_{|FV(\Delta)}$, $\mathscr{C}'\theta'\sigma \preccurlyeq \mathscr{C}$.

Proof
See Bozzano *et al.* (2003). □

The satisfiability judgment $\Vdash_\Sigma$ also satisfies the following properties.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Lemma 5
For any interpretations $I_1, I_2, \ldots$, context Δ, fact $\mathscr{C}$, and substitution θ,

i. if $I_1 \sqsubseteq I_2$ and $I_1 \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ then there exist a fact $\mathscr{C}'$, and substitutions θ' and σ s.t. $I_2 \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C}' \blacktriangleright \theta'$, $\theta_{|FV(\Delta)} = (\theta' \circ \sigma)_{|FV(\Delta)}$, $\mathscr{C}'\theta'\sigma \preccurlyeq \mathscr{C}\theta$;

ii. if $I_1 \sqsubseteq I_2 \sqsubseteq \ldots$ and $\bigsqcup_{i=1}^{\infty} I_i \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ then there exist $k \in \mathbb{N}$, a fact $\mathscr{C}'$, and substitutions θ' and σ s.t. $I_k \Vdash_\Sigma \Delta \blacktriangleright \mathscr{C}' \blacktriangleright \theta'$, $\theta_{|FV(\Delta)} = (\theta' \circ \sigma)_{|FV(\Delta)}$, $\mathscr{C}'\theta'\sigma \preccurlyeq \mathscr{C}\theta$.

Proof
See Bozzano *et al.* (2003). □

We are now ready 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 interpretations (i.e. elements of $\wp(HB(P))$). With a little bit of overloading, we will call the operator with the same name, i.e. S_P. This operator should satisfy the equation $[\![S_P(I)]\!] = T_P([\![I]\!])$ for every interpretation I. This property ensures soundness and completeness of the *symbolic* representation.

After defining the operator over $\wp(HB(P))$, we will lift it to our abstract domain $\mathscr{I}$ consisting of the equivalence classes of elements of $\wp(HB(P))$ w.r.t. the relation $\simeq$ defined in Definition 5.6. Formally, we first introduce the following definition.

Definition 5.11 (*Symbolic Fixpoint Operator* $\boldsymbol{S_P}$)
Given an $\mathrm{LO}_\forall$ program P and an interpretation I, the symbolic fixpoint operator S_P is defined as follows:

$$S_P(I) \stackrel{\text{def}}{=} \{(\widehat{H} + \mathscr{C})\,\theta \mid (H \multimap G) \in Vrn(P),\ I \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C} \blacktriangleright \theta\}.$$

Note that the S_P operator is defined using the judgment $\Vdash_{\Sigma_P}$.

Proposition 5 states that S_P is sound and complete w.r.t T_P. To prove this, we need to formulate Lemma 6 below.

Notation Let P be an $\mathrm{LO}_\forall$ program, and $\Sigma, \Sigma_1 \in Sig_P$ be two signatures such that $\Sigma_1 \subseteq \Sigma$. Given a fact $\mathscr{C}$, defined on Σ, we use $\lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1}$ to denote any fact which is obtained in the following way. For every constant (eigenvariable) $c \in (\Sigma \backslash \Sigma_1)$, pick a new variable in $\mathscr{V}$ (not appearing in $\mathscr{C}$), let it be x_c (distinct variables must be chosen for distinct eigenvariables). Now, $\lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1}$ is obtained by $\mathscr{C}$ by replacing every $c \in (\Sigma \backslash \Sigma_1)$ with x_c. For instance, if $\mathscr{C} = \{p(x, f(c)), q(y, d)\}$, with $c \in (\Sigma \backslash \Sigma_1)$ and $d \in \Sigma_1$, we have that $\lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1} = \{p(x, f(x_c)), q(y, d)\}$.

Given a context (multiset of goals) Δ, defined on Σ, we define $\lceil \Delta \rceil_{\Sigma \to \Sigma_1}$ in the same way. Similarly, given a substitution θ, defined on Σ, we use the notation $\lceil \theta \rceil_{\Sigma \to \Sigma_1}$ to denote the substitution obtained from θ by replacing every $c \in (\Sigma \backslash \Sigma_1)$ with a new variable x_c in every binding of θ. For instance, if $\theta = [u \mapsto p(x, f(c)), v \mapsto q(y, d)]$, with $c \in (\Sigma \backslash \Sigma_1)$ and $d \in \Sigma_1$, we have that $\lceil \theta \rceil_{\Sigma \to \Sigma_1} = [u \mapsto p(x, f(x_c)), v \mapsto q(y, d)]$.

Using the notation $[\![I]\!] \models_{\Sigma_1} \lceil \Delta \rceil_{\Sigma \to \Sigma_1} \blacktriangleright \lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1}$ we mean the judgment obtained by replacing every $c \in (\Sigma \backslash \Sigma_1)$ with x_c simultaneously in Δ and $\mathscr{C}$. Newly introduced variables must not appear in Δ, $\mathscr{C}$, or I.

When Σ and Σ_1 are clear from the context, we simply write $\lceil \mathscr{C} \rceil$, $\lceil \Delta \rceil$, and $\lceil \theta \rceil$ for $\lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1}$, $\lceil \Delta \rceil_{\Sigma \to \Sigma_1}$, and $\lceil \theta \rceil_{\Sigma \to \Sigma_1}$.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Finally, we use $\xi_{\Sigma_1 \to \Sigma}$ (or simply ξ if it is not ambiguous) to denote the substitution which maps every variable x_c back to c (for every $c \in (\Sigma \backslash \Sigma_1)$), i.e. consisting of all bindings of the form $x_c \mapsto c$ for every $c \in \Sigma \backslash \Sigma_1$. Clearly, we have that $\lceil F \rceil \xi = F$, for any fact or context F, and $\lceil \theta \rceil \circ \xi = \theta$ for any substitution θ.

Note that, by definition, $\lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1}$ and $\lceil \Delta \rceil_{\Sigma \to \Sigma_1}$ are defined on Σ_1, while $\xi_{\Sigma_1 \to \Sigma}$ is defined on Σ.

Lemma 6
Let P be an $\mathrm{LO}_\forall$ program, I an interpretation, and $\Sigma, \Sigma_1 \in Sig_P$ two signatures, with $\Sigma_1 \subseteq \Sigma$.

i. If $I \Vdash_{\Sigma_1} \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$ then $I \Vdash_{\Sigma} \Delta \blacktriangleright \mathscr{C} \blacktriangleright \theta$;
ii. If $[\![I]\!] \models_{\Sigma} \Delta \blacktriangleright \mathscr{C}$ then $[\![I]\!] \models_{\Sigma_1} \lceil \Delta \rceil_{\Sigma \to \Sigma_1} \blacktriangleright \lceil \mathscr{C} \rceil_{\Sigma \to \Sigma_1}$.

Proof
See (Bozzano *et al.*, 2003). □

Proposition 5
For every $\mathrm{LO}_\forall$ program P and interpretation I, $[\![S_P(I)]\!] = T_P([\![I]\!])$.

Proof
$[\![S_P(I)]\!] \subseteq T_P([\![I]\!])$.

We prove that for every $\Sigma \in Sig_P$, $[\![S_P(I)]\!]_\Sigma \subseteq T_P([\![I]\!])_\Sigma$. Assume $(\widehat{H} + \mathscr{C})\theta \in S_P(I)$, with $H \multimap G$ a variant of a clause in P and $I \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C} \blacktriangleright \theta$. Assume also that $\mathscr{A} = ((\widehat{H} + \mathscr{C})\theta + \mathscr{D})\theta' \in Inst_\Sigma(Up_\Sigma(S_P(I))) = [\![Sp(I)]\!]_\Sigma$. We have that $I \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C} \blacktriangleright \theta$ implies $I \Vdash_{\Sigma} G \blacktriangleright \mathscr{C} \blacktriangleright \theta$ by item *i* of Lemma 6 (remember that $\Sigma_P \subseteq \Sigma$). Therefore, by item *i* of Lemma 4, we get $[\![I]\!] \models_\Sigma G\theta\theta' \blacktriangleright \mathscr{C}'\theta'$ for any fact $\mathscr{C}' \succcurlyeq \mathscr{C}\theta$. Taking $\mathscr{C}' = \mathscr{C}\theta + \mathscr{D}$, it follows that $[\![I]\!] \models_\Sigma G\theta\theta' \blacktriangleright \mathscr{C}\theta\theta' + \mathscr{D}\theta'$. Therefore, by definition of T_P, we have $\widehat{H}\theta\theta' + \mathscr{C}\theta\theta' + \mathscr{D}\theta' \in (T_P([\![I]\!]))_\Sigma$, i.e. $\mathscr{A} \in (T_P([\![I]\!]))_\Sigma$.

$T_P([\![I]\!]) \subseteq [\![S_P(I)]\!]$.

We prove that for every $\Sigma \in Sig_P$, $Tp([\![I]\!])_\Sigma \subseteq [\![S_P(I)]\!]_\Sigma$. Assume $\mathscr{A} \in (T_P([\![I]\!]))_\Sigma$. By definition of T_P, there exist a variant of a clause $H \multimap G$ in P, a fact $\mathscr{C}$ and a substitution θ (defined over Σ) s.t. $\mathscr{A} = \widehat{H}\theta + \mathscr{C}$ and $[\![I]\!] \models_\Sigma G\theta \blacktriangleright \mathscr{C}$.

By item *ii* of Lemma 6 we have that $[\![I]\!] \models_\Sigma G\theta \blacktriangleright \mathscr{C}$ implies $[\![I]\!] \models_{\Sigma_P} \lceil G\theta \rceil \blacktriangleright \lceil \mathscr{C} \rceil$ (hereafter, we use the notation $\lceil \cdot \rceil$ for $\lceil \cdot \rceil_{\Sigma \to \Sigma_P}$). From $H \multimap G$ in P, we know that G is defined on Σ_P. It follows easily that $\lceil G\theta \rceil = G\lceil \theta \rceil$, so that $[\![I]\!] \models_{\Sigma_P} G\lceil \theta \rceil \blacktriangleright \lceil \mathscr{C} \rceil$. By item *ii* of Lemma 4, there exist a fact $\mathscr{C}'$, and substitutions θ' and σ (defined over Σ_P) s.t. $I \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C}' \blacktriangleright \theta'$, $\lceil \theta \rceil_{|FV(G)} = (\theta' \circ \sigma)_{|FV(G)}$, and $\mathscr{C}'\theta'\sigma \preccurlyeq \lceil \mathscr{C} \rceil$.

By definition of S_P, we have $(\widehat{H} + \mathscr{C}')\theta' \in S_P(I)$.

Now, $\mathscr{A} = \widehat{H}\theta + \mathscr{C} = \widehat{H}\lceil \theta \rceil \xi + \lceil \mathscr{C} \rceil \xi =$ (note that by hypothesis $\theta' \circ \sigma$ and $\lceil \theta \rceil$ coincide for variables in G, and are not defined on variables in H which do not appear in G because $H \multimap G$ is a variant) $\widehat{H}\theta'\sigma\xi + \lceil \mathscr{C} \rceil \xi \succcurlyeq \widehat{H}\theta'\sigma\xi + \mathscr{C}'\theta'\sigma\xi = ((\widehat{H} + \mathscr{C}')\theta')\sigma\xi \in [\![(\widehat{H} + \mathscr{C}')\theta']\!]_\Sigma \subseteq [\![S_P(I)]\!]_\Sigma$. □

The following corollary holds.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Corollary 1
For every $\mathrm{LO}_\forall$ program P and interpretations I and J, if $I \simeq J$ then $S_P(I) \simeq S_P(J)$.

Proof
If $I \simeq J$, i.e. $[\![I]\!] = [\![J]\!]$, we have that $T_P([\![I]\!]) = T_P([\![J]\!])$. By Proposition 5, it follows that $[\![S_P(I)]\!] = [\![S_P(J)]\!]$, i.e. $S_P(I) \simeq S_P(J)$. □

Corollary 1 allows us to safely lift the definition of S_P from the lattice $\langle \wp(HB(P)), \subseteq \rangle$ to $\langle \mathscr{I}, \sqsubseteq \rangle$. Formally, we define the abstract fixpoint operator as follows.

Definition 5.12 (*Abstract Fixpoint Operator* $\boldsymbol{S_P}$)
Given an $\mathrm{LO}_\forall$ program P and an equivalence class $[I]_\simeq$ of $\mathscr{I}$, the abstract fixpoint operator S_P is defined as follows:

$$S_P([I]_\simeq) \stackrel{\text{def}}{=} [S_P(I)]_\simeq$$

where $S_P(I)$ is defined in Definition 5.11.

For the sake of simplicity, in the following we will often use I to denote its class $[I]_\simeq$, and we will simply use the term *(abstract) interpretation* to refer to an equivalence class, i.e. an element of $\mathscr{I}$. The abstract fixpoint operator S_P satisfies the following property.

Proposition 6 (*Monotonicity and Continuity*)
For every $\mathrm{LO}_\forall$ program P, the abstract fixpoint operator S_P is monotonic and continuous over the lattice $\langle \mathscr{I}, \sqsubseteq \rangle$.

Proof
Monotonicity.
We prove that if $I \sqsubseteq J$, then $S_P(I) \sqsubseteq S_P(J)$, i.e. $[\![S_P(I)]\!] \subseteq [\![S_P(J)]\!]$. To prove the latter condition, we will use the characterization given by Proposition 4. Assume $\mathscr{A} = (\widehat{H} + \mathscr{C})\theta \in S_P(I)$, with $H \circ\!\!- G$ a variant of a clause in P and $I \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C} \blacktriangleright \theta$.

By item i of Lemma 5, there exist a fact $\mathscr{C}'$, and substitutions θ' and σ (note that they are defined over Σ_P) s.t. $J \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C}' \blacktriangleright \theta'$, $\theta_{|FV(G)} = (\theta' \circ \sigma)_{|FV(G)}$, $\mathscr{C}'\theta'\sigma \preccurlyeq \mathscr{C}\theta$. Let $\mathscr{C}\theta = \mathscr{C}'\theta'\sigma + \mathscr{D}$, with $\mathscr{D}$ a fact defined over Σ_P. By definition of S_P, $\mathscr{B} = (\widehat{H} + \mathscr{C}')\theta' \in S_P(J)$.

Now, $\mathscr{A} = (\widehat{H} + \mathscr{C})\theta = \widehat{H}\theta + \mathscr{C}\theta = \widehat{H}\theta'\sigma + \mathscr{C}'\theta'\sigma + \mathscr{D}$ (note in fact that by hypothesis $\theta'\sigma$ and θ coincide for variables in G, and are not defined on variables in H which do not appear in G because $H \circ\!\!- G$ is a variant). Therefore, we have that $\mathscr{A} = \widehat{H}\theta'\sigma + \mathscr{C}'\theta'\sigma + \mathscr{D} = \mathscr{B}\sigma + \mathscr{D}$.

Continuity.
We show that S_P is finitary, i.e. if $I_1 \sqsubseteq I_2 \sqsubseteq \ldots$, then $S_P(\bigsqcup_{i=1}^{\infty} I_i) \sqsubseteq \bigsqcup_{i=1}^{\infty} S_P(I_i)$, i.e. $[\![S_P(\bigsqcup_{i=1}^{\infty} I_i)]\!] \subseteq [\![\bigsqcup_{i=1}^{\infty} S_P(I_i)]\!]$. Again, we will use the characterization given by Proposition 4. Assume $\mathscr{A} = (\widehat{H} + \mathscr{C})\theta \in S_P(\bigsqcup_{i=1}^{\infty} I_i)$, with $H \circ\!\!- G$ a variant of a clause in P and $\bigsqcup_{i=1}^{\infty} I_1 \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C} \blacktriangleright \theta$.

By item *ii* of Lemma 5, there exist $k \in \mathbb{N}$, a fact $\mathscr{C}'$, and substitutions θ' and σ (note that they are defined over Σ_P) s.t. $I_k \Vdash_{\Sigma_P} G \blacktriangleright \mathscr{C}' \blacktriangleright \theta'$, $\theta_{|FV(G)} = (\theta' \circ \sigma)_{|FV(G)}$,

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$\mathscr{C}'\theta'\sigma \preccurlyeq \mathscr{C}\theta$. Let $\mathscr{C}\theta = \mathscr{C}'\theta'\sigma + \mathscr{D}$, with $\mathscr{D}$ a fact defined over Σ_P. By definition of S_P, $\mathscr{B} = (\widehat{H} + \mathscr{C}')\theta' \in S_P(I_k)$.

Exactly as above, we prove that $\mathscr{A} = (\widehat{H} + \mathscr{C})\theta = \widehat{H}\theta'\sigma + \mathscr{C}'\theta'\sigma + \mathscr{D} = \mathscr{B}\sigma + \mathscr{D}$. □

Corollary 2
For every $\mathrm{LO}_\forall$ program P, $[\![lfp(S_P)]\!] = lfp(T_P)$.

Let $\mathscr{F}_{sym}(P) = lfp(S_P)$, then we have the following main theorem.

Theorem 1 (*Soundness and Completeness*)
For every $\mathrm{LO}_\forall$ program P, $O(P) = F(P) = [\![\mathscr{F}_{sym}(P)]\!]_{\Sigma_P}$.

Proof
From Theorem 1 and Corollary 2. □

The previous results give us an algorithm to compute the operational and fixpoint semantics of a program P via the fixpoint operator S_P.

Example 5.13
Let us consider a signature with a constant symbol a, a function symbol f and predicate symbols p, q, r, s. Let $\mathscr{V}$ be a denumerable set of variables, and $u, v, w, \ldots \in \mathscr{V}$. Let us consider the program P given below.

1. $r(w) \circ\!\!- q(f(w))$
2. $s(z) \circ\!\!- \forall x.p(f(x))$
3. $\bot \circ\!\!- q(u) \,\&\, r(v)$
4. $p(x) \parr q(x) \circ\!\!- \top$

From clause 4, and using the first rule for $\Vdash_{\Sigma_P}$, we get $S_P(\emptyset) = [\{\{p(x), q(x)\}\}]_\simeq$. For simplicity, we omit the class notation, and we write

$$S_P\uparrow_1 = S_P(\emptyset) = \{\{p(x), q(x)\}\}.$$

We can now apply the remaining clauses to the element $I = \{p(x), q(x)\}$ (remember that $S_P([I]_\simeq) = [S_P(I)]_\simeq$). From the first clause (see Example 5.10) we have $I \Vdash_{\Sigma_P} q(f(w')) \blacktriangleright p(x') \blacktriangleright [x' \mapsto f(w')]$. It follows that $(r(w'), p(x'))[x' \mapsto f(w')] = r(w'), p(f(w')) \in S_P\uparrow_2$. As the reader can verify (see discussion in Example 5.10), clause 2 does not yield any further element, and the same holds for clause 3, therefore (changing w' into y for convenience)

$$S_P\uparrow_2 = \{\{p(x), q(x)\}, \{r(y), p(f(y))\}\}.$$

Now, we can apply clause 3 to the elements in $S_P\uparrow_2$. According to Example 5.10, we have that $I \Vdash_{\Sigma_P} q(u') \,\&\, r(v') \blacktriangleright p(x''') \blacktriangleright [u' \mapsto f(y'), v' \mapsto y', x''' \mapsto f(y')]$. Therefore we get that $(p(x'''))[u' \mapsto f(y'), v' \mapsto y', x''' \mapsto f(y')] = p(f(y')) \in S_P\uparrow_3$. Clause 2 cannot be applied yet, for the same reasons as above. Also, note that the element $r(y), p(f(y))$ is now subsumed by $p(f(y'))$. Therefore we can assume

$$S_P\uparrow_3 = \{\{p(x), q(x)\}, \{p(f(y'))\}\}.$$

Finally, we can apply clause 2 to $S_P\uparrow_3$, using the $\forall$-rule for the $\Vdash_{\Sigma_P}$ judgment. Take $c \notin \Sigma_P$, and consider a renaming of the last element in $S_P\uparrow_3$, $p(f(y''))$. Consider

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

(a renaming of) clause 2, $s(z') \circ\!\!- \forall x.p(f(x))$. We have that $I \Vdash_{\Sigma_P,c} p(f(c)) \blacktriangleright \epsilon \blacktriangleright nil$, with *nil* being the empty substitution. Therefore we get that $I \Vdash_{\Sigma_P} \forall x.p(f(x)) \blacktriangleright \epsilon \blacktriangleright nil$, from which $s(z') \in S_P\uparrow_4$. The reader can verify that no further clauses can be applied and that $S_P\uparrow_4$ is indeed the fixpoint of S_P, therefore we have that

$$S_P\uparrow_4 = S_P\uparrow_\omega = \{\{p(x), q(x)\}, \{p(f(y'))\}, \{s(z')\}\}.$$

Note that $F(P)$ is defined to be $[\![lfp(S_P)]\!]_{\Sigma_P}$, therefore it includes, e.g. the elements $s(a)$ (see Example 3.8), $p(f(f(y')))$ and $p(f(f(y'')))$, $q(x'')$. □

6 Ensuring termination

In general the symbolic fixpoint semantics of first order LO programs is not computable (see also the results in Cervesato *et al.* (1999)). In fact, the use of first order terms can easily lead to LO programs that encode operations over natural numbers. In this section, however, we will isolate a fragment of $LO_\forall$ for which termination of the bottom-up evaluation algorithm presented in Section 5 is guaranteed. An application of these results will be presented in Section 7. First of all, we will introduce some preliminary notions that we will use later on to prove the decidability of our fragment.

6.1 The theory of well quasi-orderings

In the following we summarize some basic definitions and results on the theory of *well quasi-orderings* (Higman, 1952; Milner, 1985; Abdulla *et al.*, 1996). A *quasi-order* $\sqsubseteq$ on a set A is a binary relation over A which is reflexive and transitive. In the following it will be denoted $(A, \sqsubseteq)$.

Definition 6.1 (*Well Quasi-Ordering*)
A quasi-order $(A, \sqsubseteq)$ is a *well quasi-ordering* (wqo) if for each infinite sequence $a_0\ a_1\ a_2\ \ldots$ of elements in A there exist indices $i < j$ such that $a_j \sqsubseteq a_i$.[1]

We have the following results, according to which a hierarchy of well quasi-orderings can be built starting from known ones. In the following $\widehat{r}$ will denote the set $\{1, \ldots, r\}$, r being a natural number, and $|w|$ the length of a string w.

Proposition 7 (*From Higman (1952)*)

i. If A is a finite set, then $(A, =)$ is a wqo;

ii. let $(A, \sqsubseteq)$ be a wqo, and let A^s denote the set of finite multisets over A. Then, $(A^s, \sqsubseteq^s)$ is a wqo, where $\sqsubseteq^s$ is the quasi-order on A^s defined as follows: given $S = \{a_1, \ldots, a_n\}$ and $S' = \{b_1, \ldots, b_r\}$, $S' \sqsubseteq^s S$ if and only if there exists an *injection* $h : \widehat{n} \to \widehat{r}$ such that $b_{h(j)} \sqsubseteq a_j$ for $1 \leqslant j \leqslant n$;

[1] Note that our $\sqsubseteq$ operator corresponds to the $\sqsupseteq$ operator of Abdulla and Jonsson (2001). Here we adhere to the classical logic programming convention.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

iii. let $(A, \sqsubseteq)$ be a wqo, and let A^* denote the set of finite strings over A. Then, $(A^*, \sqsubseteq^*)$ is a wqo, where $\sqsubseteq^*$ is the quasi-order on A^* defined in the following way: $w' \sqsubseteq^* w$ if and only if there exists a strictly monotone (meaning that $j_1 < j_2$ if and only if $h(j_1) < h(j_2)$) *injection* $h : \widehat{|w|} \to \widehat{|w'|}$ such that $w'(h(j)) \sqsubseteq w(j)$ for $1 \leqslant j \leqslant |w|$.

We are ready now to study the class of monadic $\mathrm{LO}_\forall$ specifications.

6.2 Monadic $\mathrm{LO}_\forall$ specifications

The class of specifications we are interested in consists of *monadic* predicates without function symbols. Intuitively, in this class we can represent process that carry along a single information taken from a possibly infinite domain (universal quantification introduces fresh names during a derivation).

Definition 6.2 (*Monadic $\mathrm{LO}_\forall$ Specifications*)
The class of monadic $\mathrm{LO}_\forall$ specifications consists of $\mathrm{LO}_\forall$ programs built over a signature Σ including a *finite* set of constant symbols $\mathscr{L}$, no function symbols, and a *finite* set of predicate symbols $\mathscr{P}$ with arity at most *one*.

Definition 6.3 (*Monadic Multisets and Interpretations*)
The class of monadic multisets consists of multisets of (non ground) atomic formulas over a signature Σ including a *finite* set of constant symbols $\mathscr{L}$, no function symbols, and a *finite* set of predicate symbols $\mathscr{P}$ with arity at most *one*. An interpretation consisting of monadic multisets is called a *monadic interpretation.*

Example 6.4
Let Σ be a signature including a constant symbols a, no function symbols, and predicate symbols p, q and r (with arity one), and s (with arity zero). Let $\mathscr{V}$ be a denumerable set of variables, and $x, y, \ldots \in \mathscr{V}$. Then the clause

$$p(x) \parr q(x) \parr r(x) \parr s \circ\!\!- (p(x) \parr p(a)) \,\&\, \forall v\, r(v)$$

is a monadic $\mathrm{LO}_\forall$ specification, and the multiset $\{p(x), q(y), q(x), s\}$ is a monadic multiset. □

We have the following result.

Proposition 8
The class of monadic multisets is closed under applications of S_P, i.e. for every interpretation I, if I is monadic then $S_P(I)$ is monadic.

Proof
Immediate by Definition 5.11 and Definition 5.9. □

Following Proposition 4, we define the *entailment* relation between multisets of (non ground) atomic formulas, denoted $\sqsubseteq^m$, as follows. For the sake of simplicity, in the rest of this section we will apply the following convention. Consider a monadic multiset. First of all, we can eliminate constant symbols by performing the following transformation (note that there are no other ground terms other than constants in

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

this class). For every atom $p(a)$, where p is a predicate symbol with arity one and a is a constant symbol in Σ, we introduce a new predicate symbol with arity zero, let it be p_a, and we transform the original multiset by substituting p_a in place of $p(a)$. The resulting set of predicate symbols is still finite (note that the set of constant and predicate symbols of the program is finite). It is easy to see that entailment between multisets transformed in the above way is a *sufficient* condition for entailment of the original multisets (note that the condition is not *necessary*, e.g. I cannot recognize that $p(a)$ entails $p(x)$).

Without loss of generality, we assume hereafter to deal with a set of predicate symbols with arity *one* (if it is not the case, we can complete predicate with arity less than one with *dummy* variables) and without constant symbols (otherwise, we operate the transformation previously described).

Definition 6.5
Given two multisets $\mathscr{A}$ and $\mathscr{B}$, $\mathscr{A} \sqsubseteq^m \mathscr{B}$ if and only if there exist a substitution θ and a multiset $\mathscr{C}$ such that $\mathscr{A} = \mathscr{B}\theta + \mathscr{C}$.

Then, we have the following property.

Proposition 9
If $\mathscr{A} \sqsubseteq^m \mathscr{B}$ then $[\![\mathscr{A}]\!] \subseteq [\![\mathscr{B}]\!]$.

Proof
It follows from Definition 5.6 and Proposition 4. □

Let $\mathscr{M}$ be a monadic multiset with variables $x_1, \ldots, x_k$. We define M_i as the *multiset* of predicate symbols having x_i as argument in $\mathscr{M}$, and $S(\mathscr{M})$ as the multiset $\{M_1, \ldots, M_k\}$. For instance, given the monadic multiset $\mathscr{M}$ defined as $\{p(x_1), q(x_1), p(x_1), q(x_2), r(x_2), q(x_3), r(x_3)\}$, $S(\mathscr{M})$ is the multiset consisting of the elements $M_1 = ppq$, $M_2 = qr$, and $M_3 = qr$, i.e. $S(\mathscr{M}) = \{ppq, qr, qr\}$ (where ppq denotes the multiset with two occurrences of p and one of q, and so on).

Given two multisets of multisets of predicate symbols $S = \{M_1, M_2, \ldots M_k\}$ and $T = \{N_1, N_2, \ldots, N_r\}$, let $S \sqsubseteq^s T$ if and only if there exists an *injective* mapping h from $\{1, \ldots, r\}$ to $\{1, \ldots, k\}$ such that $N_i \preccurlyeq M_{h(i)}$ for $i : 1, \ldots, r$. As an example, $\{ppp, tt, qq, rrr\} \sqsubseteq^s \{pp, q, rr\}$ by mapping: pp into ppp ($pp \preccurlyeq ppp$), q into qq ($q \preccurlyeq qq$), and rr into rrr ($rr \preccurlyeq rrr$). On the contrary, $\{ppp, rr, t, qq\} \not\sqsubseteq^s \{pq, q, rr\}$, in fact there is no multiset in the set on the left hand side of the previous relation of which pq is a sub-multiset.

The following property relates the quasi order $\sqsubseteq^s$ and the entailment relation $\sqsubseteq^m$.

Lemma 7
Let $\mathscr{M}$ and $\mathscr{N}$ be two monadic multisets. Then $S(\mathscr{M}) \sqsubseteq^s S(\mathscr{N})$ implies $\mathscr{M} \sqsubseteq^m \mathscr{N}$.

Proof
Let $S(\mathscr{M}) = \{M_1, M_2, \ldots M_k\}$ and $S(\mathscr{N}) = \{N_1, N_2, \ldots, N_r\}$. Furthermore, let h be the injective mapping from $\{1, \ldots, r\}$ to $\{1, \ldots, k\}$ such that $N_i \preccurlyeq M_{h(i)}$. By construction of M and N, it is easy to see that for every $i \in \{1, \ldots, r\}$ we can isolate atomic formulas $A_{i1}, \ldots, A_{iz}$ in $\mathscr{N}$ (corresponding to the cluster of variables N_i), where z is the cardinality of N_i, and atomic formulas $B_{i1}, \ldots, B_{iz}$ in $\mathscr{M}$ (corresponding

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

to the cluster of variables $M_{h(i)}$), such that the conditions required by Definition 6.5 are satisfied. □

As an immediate consequence of this lemma, we obtain the following property.

Proposition 10
The entailment relation $\sqsubseteq^m$ between monadic multisets is a well-quasi-ordering.

Proof
The conclusion follows from the observations below (in the following we denote by $\succcurlyeq$ the converse of the sub-multiset relation, i.e. $\mathscr{A} \succcurlyeq \mathscr{B}$ if and only if $\mathscr{B} \preccurlyeq \mathscr{A}$):

- the $\succcurlyeq$ relation is a well quasi-ordering by Dickson's Lemma (which is a consequence of Proposition 7, see also Dickson (1913)). Intuitively, multiset inclusion is equivalent to the component-wise ordering of tuples of integers denoting the occurrences of the finite set of predicate symbols in a multiset;
- since $\sqsubseteq^s$ is built over elements ordered with respect to the well quasi-ordering $\succcurlyeq$, $\sqsubseteq^s$ is in turn a well quasi-ordering by item *ii* of Proposition 7;
- as a consequence of Lemma 7, $\sqsubseteq^s$ being a well quasi-ordering implies that $\sqsubseteq^m$ is a well quasi-ordering.

□

We can now formulate the following proposition, which states that the bottom-up fixpoint semantics is computable in finite time for monadic $\mathrm{LO}_\forall$ specifications. This results relies on the following facts: in the case of monadic specifications, each interpretation computed via bottom-up evaluation consists of monadic multisets, and the entailment relation between monadic multisets is a well quasi-ordering (therefore eventually the fixpoint computation stabilizes).

Proposition 11
Let P be a monadic $\mathrm{LO}_\forall$ specification. Then there exists $k \in \mathbb{N}$ such that $\mathscr{F}_{sym}(P) = \bigsqcup_{i=0}^{k} S_P\uparrow_k(\emptyset)$.

Proof
We first note that the denotation of a monadic interpretation I is defined in terms of the denotation of its elements (monadic multisets). Thus, a monadic interpretation I represents an *upward closed* set w.r.t. to the ordering $\sqsubseteq^m$. Furthermore, the sequence of interpretations computed during a fixpoint computation forms an increasing sequence with respect to their denotation. The result follows then from Propositions 8, 9, 10, and known results on well quasi-orderings which guarantee that any infinite increasing sequence of upward-closed sets eventually stabilizes (see Finkel and Schnoebelen (2001)). □

7 An example

In this section we show how the bottom-up semantics of Section 5 can be applied for verifying the test-and-lock protocol given in Section 3.2. In order to run the experiments described hereafter, we have built a prototypical verification tool

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_\Sigma init, \top, m(a, locked), m(a, locked)}\ \top_r}{P \vdash_\Sigma init, use(a), use(a), m(a, locked), m(a, locked)}\ bc^{(8)}}{P \vdash_\Sigma init, wait(a), wait(a), m(a, unlocked), m(a, unlocked)}\ bc^{(6^*)}}{P \vdash_\Sigma init, think, think, m(a, unlocked), m(a, unlocked)}\ bc^{(4^*)}}{P \vdash_\Sigma init, think, think}\ bc^{(2^*)}}{P \vdash_\Sigma init}\ bc^{(1^*)}$$

Fig. 6. Incorrect test-and-lock protocol: a trace violating mutual exclusion.

implementing the bottom-up fixpoint procedure (backward reachability algorithm) described in Section 5. Following the guidelines and programming style described in Elliott and Pfenning (1991), we have implemented an interpreter for the relevant first order fragment of LO, enriched with the bottom-up evaluation procedure described in Section 5. The verification tool has been implemented in Standard ML. Let us consider again the test-and-lock protocol given in Section 3.2. Using our verification tool, we can now automatically verify the *mutual exclusion* property for the protocol. The specification of unsafe states is simply as follows:

$$8.\quad use(x) \parr use(x) \circ\!\!- \top$$

Note that the test-and-lock specification can be transformed into a *monadic* one. In fact, the second argument can be embedded into the predicate m so as to define the two predicates $m_{unlocked}$ and m_{locked}. In some sense, the specification is *implicitly* monadic since the second argument is defined over a finite set of states. Therefore termination of the fixpoint computation is *guaranteed* by Proposition 11. Running the verification algorithm, we actually find a mutual exclusion violation. The corresponding trace is shown in Figure 6, where $bc^{(i^*)}$ denotes multiple applications of clause number i. The problem of the above specification lies in clause 2:

$$2.\quad init \circ\!\!- init \parr m(x, unlocked)$$

In fact, using an (externally quantified) variable x does not prevent the creation of multiple monitors for the same resource. This causes a violation of mutual exclusion when different processes are allowed to concurrently access a given resource by different monitors. Figure 7 (where, for readability, we re-use the same variables in different multisets) also shows the fixpoint computed for the incorrect version of the protocol: note that the singleton multiset containing the atom *init* is in the fixpoint (this amounts to saying that there exists a state violating mutual exclusion which is reachable from the initial configuration of the protocol).

Luckily, we can fix the above problem in a very simple way. As we do not care about what resource identifiers actually are, we can elegantly encode them using universal quantification in the body of clause 2, as follows:

$$2'.\quad init \circ\!\!- init \parr \forall x.\, m(x, unlocked)$$

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$\{init\}$
$\{use(x),\ use(x)\}$
$\{m(x, unlocked),\ use(x),\ wait(y)\}$
$\{m(x, unlocked),\ use(x),\ use(y),\ m(y, locked)\}$
$\{m(x, locked),\ use(x),\ m(y, unlocked),\ m(y, unlocked),\ think\}$
$\{m(x, unlocked),\ m(x, unlocked),\ wait(y),\ think\}$
$\{m(x, unlocked),\ m(x, unlocked),\ use(y),\ m(y, locked),\ use(z),\ m(z, locked)\}$
$\{m(x, unlocked),\ m(x, unlocked),\ use(y),\ m(y, locked),\ wait(z)\}$
$\{wait(x),\ m(y, unlocked),\ m(y, unlocked),\ wait(z)\}$
$\{m(x, unlocked),\ m(x, unlocked),\ think,\ think\}$
$\{use(x),\ m(x, unlocked),\ think\}$

Fig. 7. Fixpoint computed for the incorrect test-and-lock protocol.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{P \vdash_{\Sigma,c,d} use(c), wait(d), think, m(c, locked), m(d, unlocked)}}{P \vdash_{\Sigma,c,d} wait(c), wait(d), think, m(c, unlocked), m(d, unlocked)}\, bc^{(6)}}{P \vdash_{\Sigma,c,d} wait(c), wait(d), use(c), m(c, locked), m(d, unlocked)}\, bc^{(7)}}{P \vdash_{\Sigma,c,d} wait(c), wait(d), wait(c), m(c, unlocked), m(d, unlocked)}\, bc^{(6)}}{P \vdash_{\Sigma,c,d} think, wait(d), wait(c), m(c, unlocked), m(d, unlocked)}\, bc^{(4)}}{P \vdash_{\Sigma,c,d} think, think, wait(c), m(c, unlocked), m(d, unlocked)}\, bc^{(4)}}{P \vdash_{\Sigma,c,d} think, think, think, m(c, unlocked), m(d, unlocked)}\, bc^{(4)}}{P \vdash_{\Sigma,c,d} init, think, think, think, m(c, unlocked), m(d, unlocked)}\, bc^{(3)}}{P \vdash_{\Sigma} init, think, think, think}\, bc^{(2'^{*})}}{P \vdash_{\Sigma} init}\, bc^{(1^{*})}$$

Fig. 8. A correct version of the test-and-lock protocol: example trace.

Every time a resource is created, a new constant, acting as the corresponding identifier, is created as well. Note that by the operational semantics of universal quantification, *different* resources are assigned *different* identifiers. This clearly prevents the creation of multiple monitors for the same resource. An example trace for the modified specification is shown in Figure 8 (where P is the program consisting of clauses 1, 2', 3 through 8 (see Section 3.2)).

Now, running again our verification tool on the corrected specification (termination is still guaranteed by Proposition 11), with the same set of unsafe states, we get the fixpoint shown in Figure 9. The fixpoint contains 12 elements and is reached in 7 steps. As the fixpoint does not contain *init*, mutual exclusion is verified, *for any number of processes and any number of resources.*

We conclude by showing how it is possible to optimize the fixpoint computation. Specifically, we show that it is possible to use the so called *invariant strengthening* technique in order to reduce the dimension of the sets computed during the

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$\{use(x),\ use(x)\}$
$\{m(x, unlocked),\ use(x),\ init\}$
$\{m(x, unlocked),\ use(x),\ wait(y)\}$
$\{m(x, unlocked),\ use(x),\ use(y),\ m(y, locked)\}$
$\{m(x, locked),\ use(x),\ m(y, unlocked),\ m(y, unlocked),\ think\}$
$\{m(x, unlocked),\ m(x, unlocked),\ wait(y),\ think\}$
$\{m(x, unlocked),\ m(x, unlocked),\ use(y),\ m(y, locked),\ use(z),\ m(z, locked)\}$
$\{m(x, unlocked),\ m(x, unlocked),\ use(y),\ m(y, locked),\ wait(z)\}$
$\{wait(x),\ m(y, unlocked),\ m(y, unlocked),\ wait(z)\}$
$\{m(x, unlocked),\ m(x, unlocked),\ init\}$
$\{m(x, unlocked),\ m(x, unlocked),\ think,\ think\}$
$\{use(x),\ m(x, unlocked),\ think\}$

Fig. 9. Fixpoint computed for the correct test-and-lock protocol.

$\{use(x)\ , use(x)\}$
$\{m(x, y)\ , m(x, z)\}$
$\{m(x, unlocked)\ , use(x)\ , use(y)\ , m(y, z)\}$
$\{m(x, unlocked)\ , use(x)\ , wait(y)\}$
$\{m(x, unlocked)\ , use(x)\ , init\}$
$\{use(x)\ , m(x, unlocked)\ , think\}$

Fig. 10. Fixpoint computed using invariant strengthening for the test-and-lock protocol.

fixpoint evaluation. Invariant strengthening consists of enlarging the theory under consideration with new clauses (e.g. additional clauses representing further unsafe states). We remark that this technique is perfectly sound, in the sense that if no property violations are found in the extended theory, then no violations can be found in the original one (i.e. proofs in the original theory are still proofs in the extended one).

One possibility might be to apply the so-called *counting abstraction*, i.e. turn the above $LO_\forall$ specification into a propositional program (i.e. a Petri net) by abstracting first order atoms into propositional symbols (e.g. $wait(x)$ into $wait$, and so on), and compute the structural invariants of the corresponding Petri net. However, this strategy is not helpful in this case (no meaningful invariant is found). We can still try some invariants using some *ingenuity*. For instance, consider the following invariant:

$$9.\quad m(x, y) \parr m(x, z) \multimap \top$$

For what we said previously (*different* resources are assigned *different* identifiers) this invariant must hold for our specification. Running the verification tool on this extended specification we get the fixpoint in Figure 10, containing only 6 elements and converging in 4 steps. A further optimization could be obtained by adding the invariant $use(x) \parr m(x, unlocked) \multimap \top$ (intuitively, if someone is using a given resource, the corresponding semaphore cannot be unlocked). In this case the computation converges immediately at the first step.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

8 Reachability and extensions of LO

In this paper we have focused our attention on the relationship between provability in LO and *coverability* for the configuration of a concurrent system.

Following Bozzano *et al.* (2002), to characterize *reachability* problems between two "configurations" (goal formulas) we need an extra feature of linear logic, namely the logical constant **1**. Differently from clauses with $\top$, clauses of the form $A_1 \parr \ldots \parr A_n \circ\!\!- \mathbf{1}$ make a derivation succeed if and only if the right-hand side of the current sequent matches an instance of $A_1 \parr \ldots \parr A_n$, i.e. all resources must be used in the corresponding derivation.

Going back to the notation used in Section 3.1, let P be a set of LO rewrite rules over Σ and $\mathscr{V}$, and $\mathscr{M}, \mathscr{M}'$ two multisets of ground atomic formulas (two configurations). Furthermore, let H, G the (possibly empty) $\parr$-disjunctions of ground atomic formulas such that $\widehat{H} = \mathscr{M}'$ and $\widehat{G} = \mathscr{M}$. Then, the provability of the sequent $P, H \circ\!\!- \mathbf{1} \vdash_{\mathbf{1}} G$ precisely characterizes the *reachability* of configuration $\mathscr{M}'$ from the initial configuration $\mathscr{M}$ via a sequence of multiset rewriting steps defined over the theory P (see Bozzano *et al.* (2002)). Again, this is a straightforward consequence of the properties of clauses like $H \circ\!\!- \mathbf{1}$ and of the fact that, when working with LO rewrite rules, derivations have no branching.

Example 8.1
Let us go back to Example 3.10 of Section 3.1 (compare the definitions of the formulas F_1 and F_2 given there). Let F_1' be the formula

$$p(a) \parr p(f(f(b))) \parr q(b) \parr q(b) \parr q(f(f(b))) \circ\!\!- \mathbf{1}$$

and F_2' be the formula

$$p(a) \parr q(b) \circ\!\!- \mathbf{1}$$

and $G = p(a) \parr p(b) \parr q(f(b))$. If we enrich P with F_1', instead of F_1, then we can transform the partial derivation of Figure 3 into an LO proof as shown below (where δ stands for the derivation fragment of Figure 3):

$$\dfrac{\dfrac{}{P \vdash_{\Sigma} \mathbf{1}}\ \mathbf{1}_r}{\delta}\ bc$$

The resulting LO proof also shows that from the multiset $\{p(a), p(b), q(f(b))\}$ we can reach the multiset $\{p(a), p(f(f(b))), q(b), q(b), q(f(f(b)))\}$ after a finite number of rewriting steps defined in accordance with P. Note that on the contrary (compare with Example 3.10), if we enrich P with F_2', it is *not* possible to turn the partial derivation of Figure 3 into an LO proof. In fact, every rewriting step will give us larger and larger multisets and the formula F_2' never becomes applicable. □

Particular attention must be paid to the constants introduced in a derivation. They cannot be extruded from the scope of the corresponding universal quantifier. For this reason, the formulas representing *target* configurations must be generalized by introducing universally quantified variables in place of constants

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

$$\dfrac{\dfrac{\dfrac{}{P, D \vdash_{\Sigma,c} \mathbf{1}}\ \mathbf{1}_r}{P, D \vdash_{\Sigma,c} p(f(a)), p(f(b)), q(b), q(c)}\ bc^{(D)}}{P, D \vdash_{\Sigma} p(f(a)), p(b), q(f(b))}\ bc^{(1)}$$

Fig. 11. Reachability as provability in $\mathrm{LO}_{\forall}$.

introduced in a derivation. For the sake of brevity, we will illustrate the connection between provability and reachability in the extended setting through the following example.

Example 8.2
Let Σ be the signature of Example 3.10. Let P consists of the clause

$$1.\quad p(x) \parr q(f(y)) \circ\!\!- \forall w.\,(p(f(x)) \parr q(y) \parr q(w))$$

Now, let D be the clause $\forall x.\, p(f(a)) \parr p(f(b)) \parr q(b) \parr q(x) \circ\!\!- \mathbf{1}$, and let G be the goal $p(f(a)) \parr p(b) \parr q(f(b))$. The universal quantifier is used here to generalize the representation of the target configuration. In fact, new constants will be introduced and associated to the predicate q in the derivation of the goal G. As an example, a possible derivation is shown in Figure 11 (where we have omitted applications of the $\parr_r$ rule for simplicity). The last backchaining step in Figure 11 is possible because of the universal quantifier used in D. It would not be possible to define D as $p(f(a)) \parr p(f(b)) \parr q(b) \parr q(c) \circ\!\!- \mathbf{1}$. In fact, the resulting initial sequent would violate the side condition of the $\forall_r$ proof rule that requires the freshness of the new constants introduced in a proof. □

The extension of the fixpoint semantics presented in this paper to more general linear logic languages (e.g. languages that include **1**) is a possible future direction for our research.

9 Conclusions

In this paper we have investigated the connections between techniques used for *symbolic model checking* of infinite-state systems (Abdulla *et al.*, 1996; Finkel and Schnoebelen, 2001) and provability in fragments of linear logic (Andreoli and Pareschi, 1990). The relationship between the two fields is illustrated in Figure 12. From our point of view, linear logic can be used as a unifying framework for reasoning about concurrent systems (e.g. Petri Nets, multiset rewriting, and so on). In (Bozzano *et al.*, 2002), we have applied algorithms previously developed for Petri Nets in order to derive bottom-up evaluation strategies for proposition linear logic. Conversely, in the current paper we have shown that the use of linear logic and the related bottom-up evaluation strategies can have interesting application for the automated verification of infinite-state systems in which processes are described via *colored* formulas. Several applications of the ideas presented in this paper can be found in Bozzano (2002) and Bozzano *et al.* (2002).

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Infinite State Concurrent Systems	Linear Logic Specification
transition system	LO program and proof system
transition	rule instance
current state	goal formula
initial state	initial goal
single final state	axiom with **1**
upward-closed set of states	axiom with $\top$
reachability	provability
Pre operator	T_P operator
Pre operator	$lfpT_P$

Fig. 12. Reachability versus provability.

Apart from verification purposes, the new fixpoint semantics can also be useful to study new applications of linear logic programming (e.g. for active databases as discussed in Harland and Winikoff (1998)). For this purpose, it might be interesting to extend the bottom-up evaluation framework to richer linear logic languages. Possible directions of research include languages with a richer set of connectives (e.g. Linlog (Andreoli, 1992)), or languages with more powerful type theories (e.g. LLF (Cervesato and Pfenning, 2002)).

Acknowledgements

We would like to thank the anonymous reviewers of the paper for their helpful comments.

A Notations

Multisets A multiset with elements in D is a function $\mathscr{M} : D \to \mathbb{N}$. If $d \in D$ and $\mathscr{M}$ is a multiset on D, we say that $d \in \mathscr{M}$ if and only if $\mathscr{M}(d) > 0$. For convenience, we often use the notation for sets (allowing duplicated elements) to indicate multisets, when no ambiguity arises from the context. For instance, $\{a, a, b\}$, where $a, b \in D$, denotes the multiset $\mathscr{M}$ such that $\mathscr{M}(a) = 2$, $\mathscr{M}(b) = 1$, and $\mathscr{M}(d) = 0$ for all $d \in D \setminus \{a, b\}$. Sometimes we simply write a, a, b for $\{a, a, b\}$. Finally, given a set D, $\mathscr{MS}(D)$ denotes the set of multisets with elements in D. We define the following operations on multisets. Let D be a set, $\mathscr{M}_1, \mathscr{M}_2 \in \mathscr{MS}(D)$, and $n \in \mathbb{N}$, then: ϵ is defined s.t. $\epsilon(d) = 0$ for all $d \in D$ (*empty multiset*); $(\mathscr{M}_1 + \mathscr{M}_2)(d) = \mathscr{M}_1(d) + \mathscr{M}_2(d)$ for all $d \in D$ (*union*); $(\mathscr{M}_1 \setminus \mathscr{M}_2)(d) = max\{0, \mathscr{M}_1(d) - \mathscr{M}_2(d)\}$ for all $d \in D$ (*difference*); $(\mathscr{M}_1 \cap \mathscr{M}_2)(d) = min\{\mathscr{M}_1(d), \mathscr{M}_2(d)\}$ for all $d \in D$ (*intersection*); $(n \cdot \mathscr{M})(d) = n\mathscr{M}(d)$ for all $d \in D$ (*scalar product*); $\mathscr{M}_1 \neq \mathscr{M}_2$ if and only if there exists $d \in D$ s.t. $\mathscr{M}_1(d) \neq \mathscr{M}_2(d)$ (*comparison*); $\mathscr{M}_1 \preccurlyeq \mathscr{M}_2$ if and only if $\mathscr{M}_1(d) \leqslant \mathscr{M}_2(d)$ for all $d \in D$ (*inclusion*); $(\mathscr{M}_1 \bullet \mathscr{M}_2)(d) = max\{\mathscr{M}_1(d), \mathscr{M}_2(d)\}$ for all $d \in D$ (*merge*); $|\mathscr{M}_1| = \Sigma_{d \in D} \mathscr{M}_1(d)$ (*cardinality*). We use the notation of a formal sum $\sum_{i \in I} \mathscr{M}_i$ to denote the union of a family of multisets $\mathscr{M}_i$, with $i \in I$, I being a finite set. It turns out that $(\mathscr{MS}(D), \preccurlyeq)$ has the structure of a lattice (the lattice is complete provided a greatest element is added). In particular, merge and intersection are, respectively,

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

the least upper bound and the greatest lower bound operators with respect to the multiset inclusion operator $\preccurlyeq$.

Signatures Given a set of formulas P, we denote by Σ_P the signature comprising the set of constant, function, and predicate symbols in P. We assume to have an infinite set $\mathscr{V}$ of variable symbols, usually noted x, y, z, etc. In order to deal with signature augmentation (due to the presence of universal quantification over goals) we also need an infinite set E of new constants (called *eigenvariables*). We denote by Sig_P the set of signatures which comprise at least the symbols in Σ_P (and possibly some eigenvariables).

$T_\Sigma^{\mathscr{V}}$ denotes the set of *non ground* terms over Σ, i.e. the set of terms built over $\Sigma \cup V$ where V is a denumerable set of variables. (A non ground term *may* have free variables; a *ground* term is also *non ground*).

$A_\Sigma^{\mathscr{V}}$ denotes the set of *non ground* atoms over Σ, i.e. atomic formulas built over non ground terms over Σ.

Multisets of atoms over $A_\Sigma^{\mathscr{V}}$ are also called *facts* throughout the paper, and usually noted $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$, ...

Substitutions and Multiset Unifiers We inherit the usual concept of *substitution* (mapping from variables to terms) from traditional logic programming. We always consider a denumerable set of variables $\mathscr{V}$, and substitutions are usually noted θ, σ, τ, ... We use the notation $[x \mapsto t, \ldots]$, where x is a variable and t is a term, to denote substitution bindings, with *nil* denoting the empty substitution. The application of a substitution θ to F, where F is a generic expression (e.g. a formula, a term, ...) is denoted by $F\theta$. A substitution θ is said to be *grounding* for F if $F\theta$ is ground, in this case $F\theta$ is called a *ground instance* of F. Composition of two substitutions θ and σ is denoted $\theta \circ \sigma$, e.g. $F(\theta \circ \sigma)$ stands for $(F\theta)\sigma$. We indicate the domain of a substitution θ by $Dom(\theta)$, and we say "θ defined on a signature Σ" meaning that θ can only map variables in $Dom(\theta)$ to terms in $T_\Sigma^{\mathscr{V}}$. Substitutions are ordered with respect to the ordering $\leqslant$ defined in this way: $\theta \leqslant \tau$ if and only if there exists a substitution σ s.t. $\tau = \theta \circ \sigma$. If $\theta \leqslant \tau$, θ is said to be *more general* than τ; if $\theta \leqslant \tau$ and $\tau \leqslant \theta$, θ and τ are said to be *equivalent*. Finally, $FV(F)$, for an expression F, denotes the set of *free* variables of F, and $\theta_{|W}$, where $W \subseteq \mathscr{V}$, denotes the *restriction* of θ to $Dom(\theta) \cap W$.

We need the notion of *most general unifier* (mgu). The definition of most general unifier is somewhat delicate. In particular, different classes of substitutions (e.g. idempotent substitutions) have been considered for defining most general unifiers. We refer the reader elsewhere (Eder, 1985; Lassez *et al.*, 1988; Palamidessi, 1990) for a discussion. Most general unifiers form a complete lattice with respect to the ordering $\leqslant$, provided a greatest element is added. For our purposes, we do not choose a particular class of most general unifiers, we only require the operation of *least upper bound* of two substitutions w.r.t $\leqslant$ to be defined and effective. The least upper bound of θ_1 and θ_2 is indicated $\theta_1 \uparrow \theta_2$. We refer the reader to Palamidessi (1990) for the definition of the least upper bound. The only property which we use in this paper is that $\theta_1 \leqslant (\theta_1 \uparrow \theta_2)$ and $\theta_2 \leqslant (\theta_1 \uparrow \theta_2)$, for any substitutions θ_1 and θ_2. We assume $\uparrow$ to be commutative and associative.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

We need to lift the definition of *most general unifier* from expressions to multisets of expressions. Namely, given two multisets $\mathscr{A} = \{a_1, \ldots, a_n\}$ and $\mathscr{B} = \{b_1, \ldots, b_n\}$ (note that $|\mathscr{A}| = |\mathscr{B}|$), we define a most general unifier of $\mathscr{A}$ and $\mathscr{B}$, written $mgu(\mathscr{A}, \mathscr{B})$, to be the most general unifier (defined in the usual way) of the two vectors of expressions $\langle a_1, \ldots, a_n \rangle$ and $\langle b_{i_1}, \ldots, b_{i_n} \rangle$, where $\{i_1, \ldots, i_n\}$ is a permutation of $\{1, \ldots, n\}$. Depending on the choice of the permutation, in general there is more than one way to unify two given multisets (the resulting class of mgu in general will include unifiers which are not equivalent). We use the notation $\theta = mgu(\mathscr{A}, \mathscr{B})$ to denote any unifier which is *non deterministically* picked from the set of most general unifiers of $\mathscr{A}$ and $\mathscr{B}$.

References

Abdulla, P. A., Cerāns, K., Jonsson, B. and Tsay, Y.-K. 1996. General Decidability Theorems for Infinite-State Systems. In *Proceedings 11th Annual International Symposium on Logic in Computer Science (LICS'96)*. IEEE Computer Society Press, New Brunswick, NJ, pp. 313–321.

Abdulla, P. A. and Jonsson, B. 2001. Ensuring Completeness of Symbolic Verification Methods for Infinite-State Systems. *Theoretical Computer Science 256,* 1–2, 145–167.

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

Andreoli, J.-M. and Pareschi, R. 1990. Linear Objects: Logical Processes with Built-In Inheritance. In *Proceedings 7th International Conference on Logic Programming*, D. Warren and P. Szeredi, Eds. MIT Press, Cambridge, MA, pp. 495–510.

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., Pareschi, R. and Castagnetti, T. 1997. Static Analysis 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. *Journal of Logic Programming 19-20*, 149–197.

Bozzano, M. 2002. A Logic-Based Approach to Model Checking of Parameterized and Infinite-State Systems. PhD thesis, Università di Genova.

Bozzano, M. and Delzanno, G. 2002. Automated Protocol Verification in Linear Logic. In *Proceedings 4th International Conference on Principles and Practice of Declarative Programming (PPDP'02)*. ACM Press, Pittsburgh, Pennsylvania, pp. 38–49.

Bozzano, M., Delzanno, G. and Martelli, M. 2001. An Effective Bottom-Up Semantics for First Order Linear Logic Programs. In *Proceedings 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, H. Kuchen and K. Ueda, Eds. LNCS, vol. 2024. Springer-Verlag, Tokyo, Japan, pp. 138–152.

Bozzano, M., Delzanno, G. and Martelli, M. 2002. An Effective Fixpoint Semantics for Linear Logic Programs. *Theory and Practice of Logic Programming 2,* 1, 85–122.

Bozzano, M., Delzanno, G. and Martelli, M. 2003. Model Checking Linear Logic Specifications. Extended version of the present paper with all the proofs. Available in the Computing Research Repository (CoRR), `http://xxx.lanl.gov/archive/cs/intro.html`, paper cs.PL/0309003.

Cervesato, I. 1994. Petri Nets as Multiset Rewriting Systems in a Linear Framework. Unpublished manuscript. Draft available from URL http://theory.stanford.edu/~iliano/forthcoming.html.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Cervesato, I. 1995. Petri Nets and Linear Logic: a Case Study for Logic Programming. In *Proceedings 1995 Joint Conference on Declarative Programming (GULP-PRODE'95)*, M. Alpuente and M. I. Sessa, Ed. Palladio Press, Marina di Vietri, Italy, pp. 313–318.

Cervesato, I., Durgin, N., Kanovich, M. and Scedrov, A. 2000. Interpreting Strands in Linear Logic. In *Proceedings 2000 Workshop on Formal Methods and Computer Security (FMCS'00)*, H. Veith, N. Heintze, and E. Clarke, Eds. Chicago, IL.

Cervesato, I., Durgin, N., Lincoln, P., Mitchell, J. and Scedrov, A. 1999. A Meta-notation for Protocol Analysis. In *12th Computer Security Foundations Workshop (CSFW'99)*, R. Gorrieri, Ed. IEEE Press, Mordano, Italy, pp. 55–69.

Cervesato, I. and Pfenning, F. 2002. A Linear Logical Framework. *Information and Computation 179,* 1, 19–75.

Delzanno, G. and Martelli, M. 2001. Proofs as Computations in Linear Logic. *Theoretical Computer Science 258,* 1-2, 269–297.

Dickson, L. E. 1913. Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors. *American Journal of Mathematics 35*, 413–422.

Durgin, N., Lincoln, P., Mitchell, J. and Scedrov, A. 1999. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols (FMSP'99)*, R. Gorrieri, Ed. Trento, Italy.

Eder, E. 1985. Properties of Substitutions and Unifications. *Journal of Symbolic Computation 1*, 31–46.

Elliott, C. and Pfenning, F. 1991. A Semi-Functional Implementation of a Higher-Order Logic Programming Language. In *Topics in Advanced Language Implementation*, P. Lee, Ed. MIT Press, pp. 289–325.

Engberg, U. and Winskel, G. 1990. Petri nets as models of linear logic. In *Proceedings Colloquium on Trees in Algebra and Programming*, A. Arnold, Ed. LNCS, vol. 389. Springer-Verlag, Copenhagen, Denmark, pp. 147–161.

Esparza, J., Finkel, A. and Mayr, R. 1999. On the Verification of Broadcast Protocols. In *Proceedings 14th International Symposium on Logic in Computer Science (LICS'99)*. IEEE Computer Society Press, Trento, Italy, pp. 352–359.

Esparza, J. and Melzer, S. 2000. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design 16*, 159–189.

Falaschi, M., Levi, G., Martelli, M. and Palamidessi, C. 1993. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation 103,* 1, 86–113.

Farwer, B. 1999. A Linear Logic View of Object Petri Nets. *Fundamenta Informaticae 37,* 3, 225–246.

Farwer, B. 2000. *Linear Logic Based Calculi for Object Petri Nets.* Logos Verlag. PhD thesis.

Finkel, A. 1993. The minimal coverability graph for petri nets. In *Advances in Petri Nets 1993*, G. Rozenberg, Ed. LNCS, vol. 674. Springer Verlag, pp. 210–243.

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. *Journal of Logic and Computation 5,* 2, 133–171.

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

Harland, J. and Winikoff, M. 1998. Making Logic Programs Reactive. In *Proceedings Workshop on Transactions and Change in Logic Databases (Dynamics'98)*. Manchester, UK, pp. 43–58.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Higman, G. 1952. Ordering by divisibility in abstract algebras. *Proceedings London Mathematical Society 2*, 326–336.

Hodas, J. and Miller, D. 1990. Representing Objects in a Logic Programming Language with Scoping Constructs. In *Proceedins 7th International Conference on Logic Programming*, D. H. Warren and P. Szeredi, Eds. The MIT Press, Cambridge, MA, pp. 511–526.

Jensen, K. 1997. *Coloured Petri-Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, 2 and 3*. Monographs in Theoretical Computer Science. Springer-Verlag.

Karp, R. M. and Miller, R. E. 1969. Parallel Program Schemata. *Journal of Computer and System Sciences 3,* 2, 147–195.

Kobayashi, N. and Yonezawa, A. 1994. Type-Theoretic Foundations for Concurrent Object-Oriented Programming. In *Proceedings 9th Conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA'94)*. SIGPLAN Notices, vol. 29. Portland, Oregon, pp. 31–45.

Kobayashi, N. and Yonezawa, A. 1995. Asynchronous Communication Model based on Linear Logic. *Formal Aspects of Computing 7,* 2, 113–149.

Kopylov, A. P. 1995. Decidability of Linear Affine Logic. In *Proceedings 10th Annual International Symposium on Logic in Computer Science (LICS'95)*, D. Kozen, Ed. IEEE Press, San Diego, California, pp. 496–504.

Lassez, J.-L., Maher, J. and Marriott, K. 1988. Unification Revisited. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, pp. 587–625.

Lincoln, P. 1995. Deciding provability of linear logic formulas. In *Advances in Linear Logic*. London Mathematical Society Lecture Notes Series, vol. 222. Cambridge University Press.

Lincoln, P., Mitchell, J., Scedrov, A. and Shankar, N. 1992. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic 56*, 239–311. (Also in the Proceedings of the 31th Annual Symposium on Foundations of Computer Science, St Louis, Missouri, October 1990, IEEE Computer Society Press. Also available as Technical Report SRI-CSL-90-08 from SRI International, Computer Science Laboratory.)

Lincoln, P. and Scedrov, A. 1994. First Order Linear Logic Without Modalities is NEXPTIME-Hard. *Theoretical Computer Science 135*, 139–154.

Martí-Oliet, N. and Meseguer, J. 1991. From Petri Nets to Linear Logic through Categories: A Survey. *International Journal of Foundations of Computer Science 2,* 4, 297–399.

Mayr, E. W. 1984. An Algorithm for the General Petri Net Reachability Problem . *SIAM J. Comput. 13*, 441–460.

McDowell, R., Miller, D. and Palamidessi, C. 1996. Encoding Transition Systems in Sequent Calculus. In *Proceedings Linear Logic 96 Tokyo Meeting*, J.-Y. Girard, M. Okada, and A. Scedrov, Eds. ENTCS, vol. 3. Elsevier, Keio University, Tokyo, Japan.

Meseguer, J. 1992. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science 96,* 1, 73–155.

Miller, D. 1993. The π-Calculus as a Theory in Linear Logic: Preliminary Results. In *Proceedings Workshop on Extensions of Logic Programming*, E. Lamma and P. Mello, Eds. LNCS, vol. 660. Springer-Verlag, Bologna, Italy, pp. 242–265.

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

Miller, D., Nadathur, G., Pfenning, F. and Scedrov, A. 1991. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic 51*, 125–157.

Milner, E. C. 1985. Basic wqo- and bqo-theory. In *Graphs and Orders*, I. Rival, Ed. D. Reidel Publishing Company, pp. 487–502.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press

Palamidessi, C. 1990. Algebraic properties of idempotent substitutions. In *Proceedings 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, M. S. Paterson, Ed. LNCS, vol. 443. Springer Verlag, Warwick University, UK, pp. 386–399.

Silva, M., Teruel, E. and Colom, J. 1998. Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems. In *Lectures in Petri Nets. I: Basic Models*, G. Rozenberg and W. Reisig, Eds. LNCS, vol. 1491. Springer-Verlag, pp. 309–373.

https://doi.org/10.1017/S1471068404002066 Published online by Cambridge University Press