Comparative Metric Semantics for Commit in Or-Parallel Logic Programming

Eneia Todoran
Department of Computer Science, Technical University of Cluj-Napoca
26 Baritiu Street, 3400 Cluj-Napoca, Romania
e-mail: Eneia.Todoran@cs.utcluj.ro

Jerry den Hartog and Erik de Vink
Faculty of Mathematics and Computer Science, Vrije Universiteit
De Boelelaan 1081a, NL-1081 HV Amsterdam, the Netherlands
e-mail: {jerry,vink}@cs.vu.nl

Abstract
For the control flow kernel of or-parallel Prolog with commit an operational and a denotational model are constructed and related using techniques from metric semantics. By maintaining explicit scope information a compositional handling of the commit for the denotational model is established. By application of an abstraction function, which deletes this extra information, the operational semantics is recovered.

1 Introduction

In recent years substantial progress has been reported on or-parallel logic programming systems, on successful experiments with extensive test sets performed on it, as well as on and-parallel extensions of these systems. See, e.g., [11, 13, 5, 10]. However, [7] discusses the lack of semantical consensus, at least for the Gödel language, about what should be considered as acceptable implementations of pruning operators. In general, semantical methods, in particular compositional ones, seem somewhat lacking behind in the understanding of concurrency in logic programming together with extra-logical control flow operators. Only a few references are known to us, e.g. [2, 8] to mention two of them. Paraphrasing the mantra of logic programming one can argue that concerning the semantical analysis of concurrent logic programming one has to deal with logic$*$control instead of with their sum.

In this paper we report on a comparative semantics for the control flow kernel of or-parallel logic programming with a commit operator. The comparison is made for a step-oriented operational semantics and a continuation-style denotational one. The restriction to the control flow follows the 'logic programming without logic' approach as advocated in [2]. It turns out that already in this relatively simple case intricate modeling tricks have to be applied, both for a succinct description (encoding of scope information) and for the justification of the denotational semantics (the technical aspect of using finitely non-empty and closed sets instead of non-empty compact ones). Taking all together one can not characterize this as 'a trivial exercise in semantics'. The present paper though is only a modest contribution to the enterprise of a complete compositional modeling of the class of concurrent logic programming languages with all meta-logical pruning operators that one would like to have to combine parallel systems and declarative program construction.

Starting point of our investigations are the metric techniques for comparative semantics as developed by De Bakker and co-workers (cf. [3, 4]). Main technical advantage of the usage of complete metric spaces over exploitation of complete partial orders is the existence of *unique* fixed points of contractions (Banach's Fixed Point Theorem) rather than *least* fixed points of continuous functions (Knaster-Tarski). Although the class of order-theoretical domains strictly subsumes that of metric ones, the latter provides sufficient structure for the modeling of concurrent logic programming.

Acknowledgments The work reported here was in part carried out while the first author visited Vrije Universiteit Amsterdam. We are indebted to Jaco de Bakker and his research group for their contribution to this. We are grateful to all four ILPS'97 referees for their honest opinion and their valuable comments on the earlier version of this paper.

2 Mathematical preliminaries

In this paper complete metric spaces are used as underlying mathematical structure for the semantical models. We assume known the definitions and basic facts of the following notions: complete metric space, compactness, non-expansive function and contractive function, standard metric on function spaces and products. The reader may inspect any standard textbook on metric topology or the monograph [4] for further details.

Main tool in metric semantics in general and, in particular, for the development of the operational and denotational semantics for or-parallel Prolog below, is the following classical result.

Theorem 2.1 *(Banach's Fixed Point Theorem) Let M be a complete metric space and $f\colon M \to M$ a contraction. Then f has a unique fixed point $\mathit{fix}(f)$.*

Below we will work with strings and certain sets of strings. For any alphabet $\mathcal{A}$ we let $\mathcal{A}^\infty_\delta = \mathcal{A}^* \cup \mathcal{A}^* \cdot \delta \cup \mathcal{A}^\omega$ be the collection of finite strings over $\mathcal{A}$, finite strings over $\mathcal{A}$ followed by δ and infinite strings over $\mathcal{A}$. The notation ϵ is employed to denote the empty string. We use $a \cdot x$ to denote the prefixing of a to the string x and, likewise, $a \cdot X$ for the set $\{\, a \cdot x \mid x \in X \,\}$.

Collections of strings come equipped with the Baire-distance. The Baire-distance between two strings x and y is governed by the length of their common prefix, i.e.,

$$d(x,y) \;=\; 2^{-\sup\{\, n \mid x[n]=y[n]\,\}}$$

where $x[n]$, $y[n]$ denote the prefix of x, y of length n. As key property of the Baire-distance we have $d(a \cdot x, a \cdot y) = \frac{1}{2} d(x,y)$ for all strings x, y. We have that $\mathcal{A}_\delta^\infty$ is a complete metric space. In fact, the metric on this collections is an *ultra-metric,* i.e. it satisfies the strong triangle inequality $d(x,z) \leq \max\{\, d(x,y), d(y,z) \,\}$. This property, typical for the structures modeling computational behaviors, will be needed later.

The notation $\mathcal{P}_{nco}(M)$ denotes the hyperspace of all non-empty compact subsets of a metric space M. The distance on M induces a metric on $\mathcal{P}_{nco}(M)$, the so-called Hausdorff-distance as follows:

$$\begin{aligned} & d(X,Y) \leq \varepsilon \iff \\ & \qquad \forall x \in X \exists y \in Y \colon d(x,y) \leq \varepsilon \wedge \forall y \in Y \exists x \in X \colon d(y,x) \leq \varepsilon, \end{aligned}$$

for $\varepsilon \geq 0$ and $X, Y \in \mathcal{P}_{nco}(M)$. It holds that completeness of M implies completeness of $\mathcal{P}_{nco}(M)$. If $f \colon M_1 \to M_2$ is a non-expansive function, then the 'lifting' $F \colon \mathcal{P}_{nco}(M_1) \to \mathcal{P}_{nco}(M_2)$ of the function f is defined as $F(X) = \{\, f(x) \mid x \in X \,\}$. It holds that F is well-defined, i.e. delivers non-empty and compact sets, and that F is also non-expansive. This result is called the Lifting Lemma.

Below we will employ the space $\mathcal{P}_{nco}(Act_\delta^\infty)$. For *nonempty* $X, Y \subseteq Act_\delta^\infty$ we have $d(a \cdot X, a \cdot Y) = \frac{1}{2} d(X,Y)$.

3 The language $\mathcal{L}$

In this section the abstract programming language $\mathcal{L}$ is introduced and its computational intuition is discussed. The language $\mathcal{L}$ captures the control flow of or-parallel Prolog: it provides sequential composition, both a sequential and a parallel don't know nondeterministic choice and a commit operator.

Definition 3.1 *Fix a set Act of actions and a set PVar of procedure variables, having typical elements a and x, respectively. Distinguish a special symbol* fail. *The class Stat of statements, ranged over by s, is given by the BNF*

$$s \;::=\; a \mid \mathsf{fail} \mid x \mid s \cdot s \mid s + s \mid s \oplus s \mid s{:}s.$$

The class Decl of declarations, with meta-variable D, is given by Decl = PVar → Stat and, finally, the language $\mathcal{L}$ is defined by $\mathcal{L}$ = Decl × Stat.

Actions are schematic versions of unifications. Together with procedure calls the two notions represent the usual concept of a goal. Failure is modeled in the abstract setting by the token fail. The operator '$\cdot$' is sequential conjunction, '$+$' sequential disjunction, '$\oplus$' parallel disjunction and ':' the pruning operator of commitment.

A program (D, s) of $\mathcal{L}$ consists of a declaration part D and a program body s. The declaration, conceptually a function, associates to each procedure variable a procedure body. For technical reasons we assume to have guarded recursion. However, in the setting of logic programming this is always fulfilled.

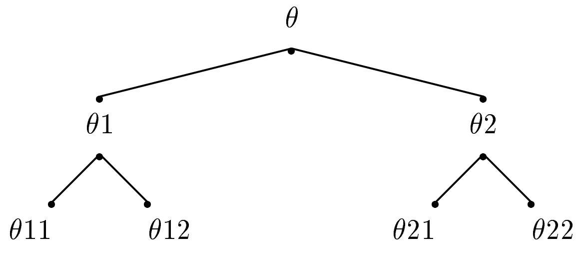
Next we discuss two problems that are encountered in the modeling of $\mathcal{L}$. The first one is caused by the presence of the don't know nondeterminism, which means that, in general, a goal $x = s_1 + s_2$ or $x = s_1 \oplus s_2$ can not be evaluated locally if it is followed by another goal x'. Rather, in executing $x \cdot x'$ first we must expand x according to its declaration and then replicate x' for each alternative of x as suggested by the following sequence of rewriting steps:

$$x \cdot x' \rightsquigarrow (s_1 + s_2) \cdot x' \rightsquigarrow (s_1 \cdot x' + s_2 \cdot x').$$

The second problem we face is how to formalize the meaning of *commit*. Intuitively, this primitive should make the current procedure deterministic, by removing the nondeterminism collected since the beginning of its execution. In this section we only explain our solution to this problem informally.

In order to define the precise scope of each commit we use a tag set *Tag*. A tag θ, σ, ρ is a non-empty sequence consisting of the natural numbers 1 and 2. Defining '$\preccurlyeq$' by $\theta \preccurlyeq \sigma$ if θ is a prefix of σ gives a partial order on tags.

The partial order $\preccurlyeq$ on *Tag* can be represented as a tree as is done in the following picture. Each node in the tree is larger then its predecessors, smaller than its successors and incomparable to nodes in other branches of the tree.



In the example above $\theta \leq \theta1 \leq \theta12$ and $\theta1$ is not comparable with $\theta2$ (i.e. neither $\theta1 \leq \theta2$ nor $\theta2 \leq \theta1$).

We will use the tags to identify the dynamic context of each recursive procedure call. Consider, for example, the following $\mathcal{L}$-programs $(D, x), (D, y)$ where x and y are the initial goals and the declaration D is given by

$$
\begin{array}{rclcrcl}
D(x) & = & (a_1 \cdot x_1) : a_2 + a_3 & \quad & D(y) & = & (a_1 \cdot y_1) \cdot y_2 + a_2 \\
D(x_1) & = & b_1 + b_2 & & D(y_1) & = & b_1 + b_2 \\
 & & & & D(y_2) & = & c_1 : c_2
\end{array}
$$

In the example above we have implicitly used that $\cdot$ and $:$ bind stronger than $+$, i.e. $s_1 \cdot s_2 + s_3 = (s_1 \cdot s_2) + s_3$. Given tags ρ for x and σ for y we label $D(x), D(y)$ as follows:

$$
\begin{array}{rcl}
D(x^\rho) & = & (a_1 \cdot x_1^{\rho 112}) :_\rho a_2 +_\rho a_3 \\
D(y^\sigma) & = & (a_1 \cdot y_1^{\sigma 112}) \cdot y_2^{\sigma 12} +_\sigma a_2
\end{array}
$$

The commit and choice operators get the same tag as the procedure in which they occur. For each recursive call in the procedure a new labels ($\rho 1$, $\rho 2$, $\rho 11$, etc.) is created such that each recursive call is receives a label larger than the label of the current procedure and incomparable with the labels of other recursive calls. This is achieved by adding a 1 or a 2 to the label depending on whether the recursive call is on the left or the right of the operator. For example the label of x_1 above is constructed as follows: First a 1 is added to ρ since x_1 is on the right of the '$+_\rho$'. Next a 1 is added since x_1 is on the right of '$:_\rho$'. Finally a 2 is added since x_1 is on the left of '$\cdot$'.

The operator '$:_\rho$' removes all the other alternatives of the choice operators '$+$' and '$\oplus$' with a tag having ρ as a prefix. This means that the commit '$:_\rho$' will exactly remove all non-deterministic alternatives created in this procedure (arguments of an operators labeled with ρ) or sub-procedures (arguments of operators labeled with a label greater then ρ). The alternatives created by other procedures will not be affected since they will have a label that is incomparable with ρ.

Below we present (possible) execution traces of the above programs. The notations are as yet informal. $s \xrightarrow{a} s'$ means that in 'state' s we can make an a-step and then continue execution in state s'. The notation $s \rightsquigarrow s'$ indicates that s is prepared for further execution by rewriting it to s'.

$$
\begin{array}{rl}
x^\rho \rightsquigarrow & (a_1 \cdot x_1^{\rho 112}) :_\rho a_2 +_\rho a_3 \\
\xrightarrow{a_1} & x_1^{\rho 112} :_\rho a_2 +_\rho a_3 \\
\rightsquigarrow & (b_1 +_{\rho 112} b_2) :_\rho a_2 +_\rho a_3 \\
\rightsquigarrow & (b_1 :_\rho a_2 +_{\rho 112} b_2 :_\rho a_2) +_\rho a_3 \\
\xrightarrow{b_1[\rho]} & a_2
\end{array}
$$

$$
\begin{array}{rl}
y^\sigma \rightsquigarrow & (a_1 \cdot y_1^{\sigma 112}) \cdot y_2^{\sigma 12} +_\sigma a_2 \\
\xrightarrow{a_1} & y_1^{\sigma 112} \cdot y_2^{\sigma 12} +_\sigma a_2 \\
\rightsquigarrow & (b_1 +_{\sigma 112} b_2) \cdot y_2^{\sigma 12} +_\sigma a_2 \\
\rightsquigarrow & (b_1 \cdot y_2^{\sigma 12} +_{\sigma 112} b_2 \cdot y_2^{\sigma 12}) +_\sigma a_2 \\
\xrightarrow{b_1} & (y_2^{\sigma 12} +_{\sigma 112} b_2 \cdot y_2^{\sigma 12}) +_\sigma a_2 \\
\rightsquigarrow & (c_1 :_{\sigma 12} c_2 +_{\sigma 112} b_2 \cdot y_2^{\sigma 12}) +_\sigma a_2 \\
\xrightarrow{c_1[\sigma 12]} & (c_2 +_{\sigma 112} b_2 \cdot y_2^{\sigma 12}) +_\sigma a_2
\end{array}
$$

The first four steps are similar in both cases. The first step is expanding a procedure according to its definition. The second step is taking an action. The third step is again expanding a procedure and the fourth step is replication for each alternative as explained above.

The next step for x is the action b_1 after which we must commit to the current alternative. This is denoted by $b_1[\rho]$. As described before the alternatives that have to be removed are those with labels which have prefix ρ. This means that both other alternatives are removed since the labels of the operators are $\rho 112$ and ρ.

For y the next steps are b_1 and then again expanding a procedure. The final step given is $c_1[\sigma 12]$. In this case no alternative is removed since the neither $\sigma 112$ nor σ has $\sigma 12$ as a prefix.

4 Operational semantics

In this section the computational intuition behind the examples discussed above is formally described using a labeled transition system. From the transition system an operational semantics is derived in the standard way. Main technicality concerns the encoding of the scope of the commit operator.

We define the set $Tag' = Tag \cup \{\infty\}$. The special symbol ∞ will be used to label actions that do not commit. Note that (since $\infty \notin Tag$) we have $\neg(\infty \preccurlyeq \rho)$ for any $\rho \in Tag$. ϱ will range over Tag'.

Definition 4.1 *The collection Res of (syntactic) resumptions with typical element r, and the collection Conf of configurations, ranged over by t, are given as follows:*

$$r ::= \mathrm{E} \mid \langle s, \theta, \sigma \rangle :_\varrho r \quad \textit{and} \quad t ::= r \mid t_1 +_\rho t_2 \mid t_1 \oplus_\rho t_2 \, .$$

The collection of tagged actions, with meta-variable α, is simply $Act \times Tag'$. We write $a[\varrho]$ for a pair (a, ϱ) in $Act \times Tag'$ and put $tag(a[\varrho]) = \varrho$.

The special symbol E denotes proper termination. Generally, resumptions are sequences of triples of the form $\langle s, \theta, \sigma \rangle$. The basic idea is that $\langle s, \theta, \sigma \rangle :_\varrho r$ starts its computation with executing $\langle s, \theta, \sigma \rangle$. After having finished this it continues the computation with that of r. A triple $\langle s, \theta, \sigma \rangle$ consists of a statement $s \in Stat$ and two tags $\theta, \sigma \in Tag$. In the definition of the transition system below we will use θ to identify the context of the current procedure, and σ to generate fresh tags for each recursive procedure call.

The token '$:_\rho$' is used to model a commit with scope ρ. If an action is executed it complies to the scope information and results, in case ρ is the current context, in a tagged action $a[\rho]$. The tag on the action can then be employed to kill possible other alternatives, thus having a net effect of a commitment. The token '$:_\infty$' is used to model the and ('$\cdot$') operator. Actions labeled with ∞ take the place of the unlabeled actions used in the previous section. Configurations are either simple resumptions or composite structures defined by means of the binary operator symbols '$+_\rho$' and '$\oplus_\rho$' that also incorporate some appropriate scoping information.

The computation steps for $\mathcal{L}$ are given by a transition system, i.e. a relation defined by axioms and rules, on $Conf \times Act \times Tag' \times Conf$. For clarity of presentation we will suppress the global declaration part of a program by assuming some fixed declaration D (instead of working with pairs of declarations and configurations).

Definition 4.2 *The transition system for $\mathcal{L}$ is given by the following axiom and rules:*

- $\langle a, \theta, \sigma \rangle :_{\varrho} r \xrightarrow{a[\varrho]} r$ *(Act)*

- $$\frac{\langle D(x), \sigma, \sigma \rangle :_{\varrho} r \xrightarrow{\alpha} t}{\langle x, \theta, \sigma \rangle :_{\varrho} r \xrightarrow{\alpha} t}$$ *(Rec)*

- $$\frac{\langle s_1, \theta, \sigma 1 \rangle :_{\infty} (\langle s_2, \theta, \sigma 2 \rangle :_{\varrho} r) \xrightarrow{\alpha} t}{\langle s_1 \cdot s_2, \theta, \sigma \rangle :_{\varrho} r \xrightarrow{\alpha} t}$$ *(And)*

- $$\frac{(\langle s_1, \theta, \sigma 1 \rangle :_{\varrho} r) \ \mathsf{op}_{\theta} \ (\langle s_2, \theta, \sigma 2 \rangle :_{\varrho} r) \xrightarrow{\alpha} t}{\langle s_1 \ \mathsf{op} \ s_2, \theta, \sigma \rangle :_{\varrho} r \xrightarrow{\alpha} t} \quad \mathsf{op} \in \{+, \oplus\}$$ *(Op)*

- $$\frac{\langle s_1, \theta, \sigma 1 \rangle :_{\theta} (\langle s_2, \theta, \sigma 2 \rangle :_{\varrho} r) \xrightarrow{\alpha} t}{\langle s_1 : s_2, \theta, \sigma \rangle :_{\varrho} r \xrightarrow{\alpha} t}$$ *(Commit)*

- $$\frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 +_{\rho} t_2 \xrightarrow{\alpha} t_1'} \quad \textit{if } tag(\alpha) \preccurlyeq \rho$$ *(SeqOr 1)*

  $$\frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 +_{\rho} t_2 \xrightarrow{\alpha} t_1' +_{\rho} t_2} \quad \textit{if } \neg(tag(\alpha) \preccurlyeq \rho)$$ *(SeqOr 2)*

  $$\frac{t_1 \not\rightarrow \quad t_2 \xrightarrow{\alpha} t_2'}{t_1 +_{\rho} t_2 \xrightarrow{\alpha} t_2'}$$ *(SeqOr 3)*

- $$\frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 \oplus_{\rho} t_2 \xrightarrow{\alpha} t_1'} \quad \textit{if } tag(\alpha) \preccurlyeq \rho$$ *(ParOr 1)*

  $$\frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 \oplus_{\rho} t_2 \xrightarrow{\alpha} t_1' \oplus_{\rho} t_2} \quad \textit{if } \neg(tag(\alpha) \preccurlyeq \rho)$$ *(ParOr 2)*

  $$\frac{t_1 \xrightarrow{\alpha} t_1' \quad t_2 \not\rightarrow}{t_1 \oplus_{\rho} t_2 \xrightarrow{\alpha} t_1'}$$ *(ParOr 3)*

  three symmetric rules *(ParOr 4,5,6)*

In the axiom (Act), the action a is augmented with the scope information of the commit, in this case with ϱ. This way the action turns into a committing action. Note that there is no axiom or rule available for fail. The rule (Rec) is basically the usual copy rule which embodies handling of recursion by body replacement. In addition, the context information is updated, the tag σ replaces the tag θ. In the rules (And), (Op) and (Commit) new tags $\sigma 1$ and $\sigma 2$ are generated, so that the recursive calls in s_1 and s_2 will be executed in different contexts. Each commit is adorned with a tag that identifies the context of the current goal. In the (Op)-rule the resumption r is replicated for each alternative of the leftmost goal.

The notation $t_1 \not\rightarrow$ in the sets of rules (SeqOr) and (ParOr) means that t_1 has no transitions, i.e. there is no (tagged) action α and no configuration t_1' such that $t_1 \xrightarrow{\alpha} t_1'$. In case t_1 or t_2 has no transitions it is dropped from the configuration providing other options are available. The rules (SeqOr 1) and (ParOr 1,4) capture, in fact, the behavior of the commit. If one of the alternatives successfully performs a tagged action $a[\theta]$ then the computation commits to this choice. Therefore other possibilities will be discarded. These are exactly generated by the subconfigurations in the scope of '$+_\rho$' and '$\oplus_\rho$' for which $\theta \preccurlyeq \rho$. In case ρ is independent from θ, i.e. $\neg(\theta \preccurlyeq \rho)$, the execution of the action induces no commitment for the or-parallelism at level ρ and the alternative remains available. Since ∞ is incomparable with any tag ρ an action with label ∞ will not commit. Note that, as is to be expected, the (SeqOr)-rules are asymmetric in t_1 and t_2. Transitions possible for t_2 only get propagated to $t_1 +_\rho t_2$ provided t_1 itself fails, i.e. has no transitions.

As an aside, it could in principle be the case that the transition system as given above would not be well-defined, due to the presence of so-called negative premises, viz. the conditions of the format $t \not\rightarrow$. However, by stratification techniques borrowed from the model-theoretic semantics for logic programming with negation, one can assure that the underlying set operator —corresponding to the T-operator— is monotone and that therefore a smallest subset satisfying the axiom and rules, i.e. a least fixed point, exists. See, e.g., [6].

One can define a so-called complexity measure for configurations such that the premises of the rules are less complex than their conclusions. (See, e.g., [2] or [4] for more details.) This implies strong normalization of the transition system as a deductive theory. More importantly though, it provides us with an induction principle that we refer to as induction on complexity measure. A straightforward application of this principle amounts to the next result. (In the comparison of the operational and denotational semantics the principle will be used again.)

Lemma 4.3 *The transition relation '$\longrightarrow$', as given in Definition 4.1, is finitely branching, i.e. it holds that $\{ (\alpha, t') \mid t \xrightarrow{\alpha} t' \}$ is a finite set, for all configurations $t \in Conf$.*

The operational semantics $\mathcal{O}[\![\cdot]\!]$ for $\mathcal{L}$ simply collects all the finite and infinite completed computations for a configuration. For convenience we employ an auxiliary mapping $\mathcal{O}$. The model $\mathcal{O}[\![\cdot]\!]$ is obtained from this $\mathcal{O}$ by initializing the proper tags and lifting a statement to a resumption.

Definition 4.4 *The semantical mapping $\mathcal{O}: Conf \to \mathcal{P}(Act_\delta^\infty)$ is given by $\mathcal{O}(\mathrm{E}) = \{\epsilon\}$ and for $t \neq E$*

$$\mathcal{O}(t) = \begin{cases} \{\delta\} & \text{if } t \not\to \\ \bigcup\{\, a \cdot \mathcal{O}(t') \mid \exists \varrho \colon t \xrightarrow{a[\varrho]} t' \,\} & \text{otherwise.} \end{cases}$$

The operational semantics $\mathcal{O}[\![\cdot]\!]: \mathcal{L} \to \mathcal{P}(Act_\delta^\infty)$ is defined as

$$\mathcal{O}[\![s]\!] = \mathcal{O}(\langle s, 1, 1\rangle :_\infty \mathrm{E}).$$

Note that the above definition is reflexive. A standard way in metric semantics to overcome this is by characterization of the object to be defined as the fixed point of a suitable contraction on a complete metric space using Banach's theorem. For $\mathcal{O}$ above we have the following fixed point description.

Theorem 4.5 *Let $Sem = Conf \to \mathcal{P}_{nco}(Act_\delta^\infty)$. Define the transformation $\Phi: Sem \to Sem$ by $\Phi(S)(\mathrm{E}) = \{\epsilon\}$ and for $t \neq \mathrm{E}$*

$$\Phi(S)(t) = \begin{cases} \{\delta\} & \text{if } t \not\to \\ \bigcup\{\, a \cdot S(t') \mid \exists \varrho \colon t \xrightarrow{a[\varrho]} t' \,\} & \text{otherwise,} \end{cases}$$

for all $S \in Sem$. Then it holds that Φ is a contraction and $\mathcal{O} = \mathit{fix}(\Phi)$.

5 Denotational semantics

The next step is the development of the denotational semantics for $\mathcal{L}$. For a correct handling of the commit the semantical operator corresponding to sequential and parallel disjunction are parameterized with scope information. Justification of the various definitions can be obtained using appropriate fixed point characterizations.

Definition 5.1 *Let the complete metric spaces $\mathbb{P}$ and $\mathbb{Q}$ be given by $\mathbb{P} = \mathcal{P}_{nco}(\mathbb{Q})$ and $\mathbb{Q} = (Act \times Tag')_\delta^\infty$. We use p and q to range over $\mathbb{P}$ and $\mathbb{Q}$, respectively. Define, for $\rho \in Tag$, the semantical operator $+_\rho: \mathbb{P} \times \mathbb{P} \to \mathbb{P}$ by*

$$p +_\rho p' = \{\, q +'_\rho q' \mid q \in p, q' \in p' \,\}$$

where '$+'_\rho$' is given as $\epsilon +'_\rho q' = q'$, $\delta +'_\rho q' = q'$, $(\alpha \cdot q) +'_\rho q' = \alpha \cdot q$ if $tag(\alpha) \preccurlyeq \rho$, and $(\alpha \cdot q) +'_\rho q' = \alpha \cdot (q +'_\rho q')$ otherwise. The semantical operator $\oplus_\rho: \mathbb{P} \times \mathbb{P} \to \mathbb{P}$, for $\rho \in Tag$, is given by

$$p \oplus_\rho p' = (p +_\rho p') \cup (p' +_\rho p).$$

The above definitions can be justified using so-called higher-order operations and by application of the Lifting Lemma. One can additionally show that the resulting fixed point, i.e. the operators $+_\rho$ and $\oplus_\rho$ on $\mathbb{P}$, are non-expansive. A property that we will need for the justification of the definition of the denotational semantics. Note that we do not define semantical counterparts of sequential conjunction and the commit itself. These constructions will be modeled using the tag-information explicitly.

The compositional model that we propose for $\mathcal{L}$ is a continuation semantics. Continuations $\gamma \in Cont$, which in fact coincide with the processes $p \in \mathbb{P}$, represent the further behavior of a process after completion of the part pertaining to the execution of the particular statement. Scope information θ, σ, ρ concerning the commit, can be easily propagated as they are arguments of the semantical mapping.

Definition 5.2 *Let Cont, ranged over by γ, be the collection of continuations given by Cont $= \mathbb{P}$. Let Tags be short for Tag $\times$ Tag $\times$ Tag'. The semantical mapping $\mathcal{D}$: Stat $\to$ Cont $\to$ Tags $\to \mathbb{P}$ is given by*

$$\begin{aligned}
\mathcal{D}(a)(\gamma) &= \lambda(\theta,\sigma,\varrho).\ a[\varrho]\cdot\gamma \\
\mathcal{D}(\mathsf{fail})(\gamma) &= \lambda(\theta,\sigma,\varrho).\ \{\delta\} \\
\mathcal{D}(x)(\gamma) &= \lambda(\theta,\sigma,\varrho).\ \mathcal{D}(D(x))(\gamma)(\sigma)(\sigma)(\varrho) \\
\mathcal{D}(s_1\cdot s_2)(\gamma) &= \lambda(\theta,\sigma,\varrho).\ \mathcal{D}(s_1)(\mathcal{D}(s_2)(\gamma)(\theta,\sigma 2,\varrho))(\theta,\sigma 1,\infty) \\
\mathcal{D}(s_1+s_2)(\gamma) &= \lambda(\theta,\sigma,\varrho).\ (\mathcal{D}(s_1)(\gamma)(\theta,\sigma 1,\varrho)) +_\theta (\mathcal{D}(s_1)(\gamma)(\theta,\sigma 2,\varrho)) \\
\mathcal{D}(s_1\oplus s_2)(\gamma) &= \lambda(\theta,\sigma,\varrho).\ (\mathcal{D}(s_1)(\gamma)(\theta,\sigma 1,\varrho)) \oplus_\theta (\mathcal{D}(s_1)(\gamma)(\theta,\sigma 2,\varrho)) \\
\mathcal{D}(s_1{:}s_2)(\gamma) &= \lambda(\theta,\sigma,\varrho).\ \mathcal{D}(s_1)(\mathcal{D}(s_2)(\gamma)(\theta,\sigma 2,\varrho))(\theta,\sigma 1,\theta).
\end{aligned}$$

The denotational semantics $\mathcal{D}[\![\cdot]\!]: \mathcal{L} \to \mathbb{P}$ is given by

$$\mathcal{D}[\![s]\!] \quad = \quad \mathcal{D}(s)(\{\epsilon\})(1,1,\infty).$$

The meaning of an action a is the action itself together with the scoping information of a possible commitment with respect to this action. Since the model is compositional we have to take the possibility of evaluation of a in a context of a sequential or parallel disjunction into account. After $a[\varrho]$ the process continues with the behavior encoded in the continuation γ. Syntactic failure results in semantic failure independent of the continuation. Recursion is here also modeled by body replacement. Note that the right-hand side is syntactically not simpler than the left-hand side. An additional argument based on a suitable complexity measure can handle this.

The sequential conjunction amounts in the update of the continuation, since the computation for the second component has to be performed after the one for the first component has finished. The clause for the commit is similar. The two definitions differ in the way the scoping information is dealt with. For the sequential case the ρ set to ∞, since a commitment made for s_1 will have no effect on the search space for s_2, whereas it does for the case of a commit. The sequential and parallel disjunction are treated by the semantical operators given in definition 5.1 above.

Well-definedness of the function $\mathcal{D}$ can be obtained by characterizing $\mathcal{D}$ as a fixed point of a higher-order transformation on a suitable subspace of $Stat \to Cont \to Tags \to \mathbb{P}$. In order to show contractivity of the transformation one uses non-expansiveness of the semantical operators, contractivity of the transformation in the continuation and ultra-metricity of the distance of $\mathbb{P}$. (See for a similar argument, e.g., [1, 4].)

6 Relating $\mathcal{O}$ and $\mathcal{D}$

Having defined both an operational and a denotational semantics the question about there relationship will be addressed in this section. A compositional treatment forced us to deal with explicit scope information concerning the commit for the denotational semantics. In the step-oriented operational model there seems no point in delivering this tags to the meaning of a program. We will argue that the operational and denotational semantics coincide once the superfluous scope information has been removed from the latter. The proof of this takes advantage of the metric machinery, in particular the uniqueness of fixed point, underlying the semantical modeling of the paper.

First of all we will introduce an operational semantical mapping $\mathcal{O}^*$ that, in contrast to $\mathcal{O}$, yields outcomes in $\mathcal{P}_{nco}((Act \times Tag')^{\infty}_{\delta})$, as is the case for the denotational semantics. The function $\mathcal{O}^*$ acts on configurations and is given as the fixed point of a transformation Φ^*. Also an abstraction function abs is given which deletes the tags from strings over $Act \times Tag$ resulting in strings over Act only.

On the one hand, it is shown that $\mathcal{O} = abs \circ \mathcal{O}^*$ employing the fixed point characterization of $\mathcal{O}$ of Section 4. On the other hand, the denotational semantics can be massaged to a mapping $\mathcal{D}^*$ on configurations (as is the case for $\mathcal{O}^*$). It is claimed that $\mathcal{D}^*$ is a fixed point of Φ^*, which, by definition, equals $\mathcal{O}^*$. So $\mathcal{O}^* = \mathcal{D}^*$. Combining the two result yields $\mathcal{O}[\![\cdot]\!] = abs \circ \mathcal{D}[\![\cdot]\!]$ for programs of $\mathcal{L}$. This is exactly the contents of Theorem 6.4.

Definition 6.1 *Put* $Sem = Conf \to \mathcal{P}_{nco}((Act \times Tag')^{\infty}_{\delta})$. *Define* $\Phi^* \colon Sem \to Sem$ *by* $\Phi^*(S)(\mathrm{E}) = \{\epsilon\}$ *and*

$$\Phi^*(S)(t) \quad = \quad \begin{cases} \{\delta\} & \textit{if } t \not\rightarrow \\ \bigcup\{\, \alpha \cdot S(t') \mid t \xrightarrow{\alpha} t' \,\} & \textit{otherwise,} \end{cases}$$

for all $S \in Sem$. *Let* $\mathcal{O}^* = \mathit{fix}(\Phi^*)$.
The abstraction function $\mathrm{abs} \colon \mathcal{P}_{nco}((Act \times Tag')^{\infty}_{\delta}) \to \mathcal{P}_{nco}(Act^{\infty}_{\delta})$ *is given by* $abs(p) = \{\, abs'(q) \mid q \in p \,\}$ *where* $abs'(q)$ *is given by* $abs'(\epsilon) = \epsilon$, $abs'(\delta) = \delta$, *and* $abs'(a[\varrho] \cdot q) = a \cdot abs'(q)$.
The semantical mapping $\mathcal{D}^* \colon Conf \to \mathcal{P}_{nco}((Act \times Tag')^{\infty}_{\delta})$ *is defined as*

$$\begin{aligned} \mathcal{D}^*(\mathrm{E}) \quad &= \quad \{\epsilon\} \\ \mathcal{D}^*(\langle s, \theta, \sigma \rangle :_{\varrho} r) \quad &= \quad \mathcal{D}(s)(\mathcal{D}^*(r))(\theta, \sigma, \varrho) \\ \mathcal{D}^*(t_1 \ \mathsf{op}_{\theta} \ t_2) \quad &= \quad \mathcal{D}^*(t_1) \ \mathsf{op}_{\theta} \ \mathcal{D}^*(t_2) \qquad \mathsf{op} \in \{+, \oplus\}. \end{aligned}$$

A technical complication arises with the justification of the above definition for the case of the abstraction function. To the other cases the standard techniques discussed earlier apply straightforwardly. The point concerning abs is that there is, in general, no guarantee that a compact sets of strings over $Act \times Tag$ remains compact when removing the tags, since the extra pieces of information separate tagged actions that are identified in Act. However, the methods available for finitary non-empty and closed sets can be used here instead (cf. [12, 4]).

The first building block for the comparison of $\mathcal{O}$ and $\mathcal{D}$ is the following.

Lemma 6.2 *$\mathcal{O} = abs \circ \mathcal{O}^*$.*

The lemma follows from the observation that the function $abs \circ \mathcal{O}^*$ is a fixed point of the transformation Φ of Theorem 4.5. Loosely speaking, the definition of Φ and Φ^* are the same modulo abstraction and the function abs behaves properly concerning set union. As a consequence of Theorem 4.5 we then have that $abs \circ \mathcal{O}^*$ coincides with $\mathcal{O}$ since the latter is the *unique* fixed point of the transformation Φ.

The second ingredient for establishing a relationship between $\mathcal{O}$ and $\mathcal{D}$ focuses on the equality of $\mathcal{O}^*$ and $\mathcal{D}^*$. We will sketch only one typical case in the proof of $\Phi^*(\mathcal{D}^*) = \mathcal{D}^*$ which, in general, goes by induction on a suitable complexity measure on configurations (as discussed also in Section 4). From the lemma it follows that $\mathcal{O}^* = \mathcal{D}^*$, again by uniqueness of fixed points.

Lemma 6.3 *$\Phi^*(\mathcal{D}^*) = \mathcal{D}^*$.*

Proof One shows, by induction on the complexity of a configuration, $\Phi^*(\mathcal{D}^*)(t) = \mathcal{D}^*(t)$. We only provide the case for $t \equiv t_1 +_\rho t_2$ where t_1 has one or more transitions:

$$
\begin{aligned}
&\Phi^*(\mathcal{D}^*)(t_1 +_\rho t_2)\\
&\quad = [\text{inspection transition system }]\\
&\qquad \textstyle\bigcup\{\, \mathcal{D}^*(t_1') \mid t_1 \xrightarrow{\alpha} t_1' \wedge tag(\alpha) \preccurlyeq \rho \,\} \cup\\
&\qquad \textstyle\bigcup\{\, \mathcal{D}^*(t_1' +_\rho t_2) \mid t_1 \xrightarrow{\alpha} t_1' \wedge \neg(tag(\alpha) \preccurlyeq \rho) \,\}\\
&\quad = [\text{property } \mathcal{D}^* \,]\\
&\qquad \textstyle\bigcup\{\, \mathcal{D}^*(t_1') \mid t_1 \xrightarrow{\alpha} t_1' \wedge tag(\alpha) \preccurlyeq \rho \,\} \cup\\
&\qquad (\textstyle\bigcup\{\, \mathcal{D}^*(t_1') \mid t_1 \xrightarrow{\alpha} t_1' \wedge \neg(tag(\alpha) \preccurlyeq \rho) \,\} +_\rho \mathcal{D}^*(t_2))\\
&\quad = [\text{definition } \Phi^*, \text{ not } t_1 \not\rightarrow]\ \ \Phi^*(\mathcal{D}^*)(t_1) +_\rho \mathcal{D}^*(t_2)\\
&\quad = [\text{induction hypothesis}]\ \ \mathcal{D}^*(t_1) +_\rho \mathcal{D}^*(t_2)\\
&\quad = [\text{definition } \mathcal{D}^*]\ \ \mathcal{D}^*(t_1 +_\rho t_2).
\end{aligned}
$$

□

We are now in a position to prove the main result of this section.

Theorem 6.4 $\mathcal{O}[\![(D,s)]\!] = abs(\mathcal{D}[\![(D,s)]\!])$ *for all programs* $(D,s) \in \mathcal{L}$.

Proof Suppressing again the declaration part D we have

$$
\begin{aligned}
\mathcal{O}[\![s]\!] &= [\text{definition } \mathcal{O}[\![\cdot]\!]] \quad \mathcal{O}(\langle s,1,1,\rangle :_\infty \mathrm{E})] \\
&= [\text{by Lemma 6.2 and 6.3}] \quad abs(\mathcal{D}^*(\langle s,1,1\rangle :_\infty \mathrm{E})) \\
&= [\text{definition of } \mathcal{D}^* \text{ and } \mathcal{D}] \quad abs(\mathcal{D}(s)(\{\epsilon\})(1,1,\infty)) \\
&= [\text{definition } \mathcal{D}[\![\cdot]\!]] \quad abs(\mathcal{D}[\![s]\!]). \qquad \square
\end{aligned}
$$

Since we have hard-wired the scoping information into the denotational semantics —in particular concerning the semantical conjunction operators— we do not have a full-abstractness result for the compositional model. In fact, an easy example shows that two programs may behave the same in any context but are still being distinguished by the denotational semantics. It would be interesting to see how equality modulo scope encoding can be defined, and, subsequently, what relative full-abstractness result holds for the proposed models.

7 Concluding remarks

By using a suitable encoding of the scope information concerning the commit operator, an operational and a denotational semantics for or-parallel logic programming with commit can be constructed. The operational model is based on a transition system which captures the computational intuition; the denotational model takes compositionality as a starting point and resorts to fixed point arguments for its definition. In order to achieve compositionality for the denotational model scope information has to be maintained explicitly. By systematic use of Banach's fixed point theorem, as is available in the metric set-up of the paper, one proves a correctness result for the compositional model $\mathcal{D}$ relative to the transitional model $\mathcal{O}$ modulo a suitable abstraction function. The paper thus shows that the control flow of or-parallel Prolog with the meta-logical operation of commitment can be modeled using comparative metric semantics.

A question spawned of from the research reported here was triggered by the wish to incorporate and-parallelism into the models. The technical issue to be addressed concerns the combination, for the denotational semantics, of continuations and explicit parallelism. In current work of the third author with Franck van Breugel, in a setting of imperative-style distributed programming, a solution for this matter is studied. The result obtained there will help in paving the way for a compositional treatment of the control flow of and/or logic programming and associated pruning operators.

Logic programming carries its own promises and, consequently, its own focus in research interests. However, various proposals made in the context of logic programming go beyond the arbitrarily risen boarders of the field. E.g., Gregory's notion of 'wait_idle' given in [9] for speculative parallelism in Parlog may very well be interpreted for general concurrent programming. The distinguishing mixture of declarative and implementation oriented reasoning makes the concept 'in between' the programming and operating system level, as it abstracts away from the underlying machine architecture. At present analysis of such a notion seems out of reach of present-day compositional methods. The 'location' process algebras as advocated, e.g., by [14] go in part into this direction. It would be interesting to see if denotational methods, in particular the metric one, can be developed to capture this kind of constructs.

References

[1] P.H.M. America and J.W. de Bakker. Designing equivalent semantic models for process creation. *Theoretical Computer Science*, 60:109–176, 1988.

[2] J.W. de Bakker. Comparative semantics for flow of control in logic programming without logic. *Information and Computation*, 94:123–179, 1991.

[3] J.W. de Bakker and J.J.M.M. Rutten, editors. *Ten Years of Concurrency Semantics, selected papers of the Amsterdam Concurrency Group.* World Scientific, 1992.

[4] J.W. de Bakker and E.P. de Vink. *Control Flow Semantics.* Foundations of Computing Series. The MIT Press, 1996.

[5] A. Beaumont and D.H.D. Warren. Scheduling speculative work in or-parallel Prolog systems. In *Proc. ICLP'93*, pages 135–149. The MIT Press, 1993.

[6] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. In *Proc. ICALP'91*, pages 481–494. LNCS 510, 1991.

[7] A. Brogi and C. Guarino. Pruning the search space of logic programs. In *Proc. ELP'96*, pages 35–49. LNAI 1050, 1996.

[8] A. Eliëns. *DLP: A Language for Distributed Logic Programming: Design, Semantics and Implementation.* Wiley, 1992.

[9] S. Gregory. Experiments with speculative parallelism in Parlog. In *Proc. ILPS'93*, pages 370–387. The MIT Press, 1993.

[10] G. Gupta and V. Santos Costa. Cuts and side-effects in and-or parallel Prolog. *Journal of Logic Programming*, 27:45–71, 1996.

[11] B. Hausman. *Pruning and Speculative Work in Or-Parallel Prolog.* PhD thesis, Royal Institute of Technology, Stockholm, 1990.

[12] E. Horita. A fully abstract model for a nonuniform concurrent language with parametrization and locality. In *Semantics: Foundations and Applications*, pages 288–317. LNCS 666, 1993.

[13] R. Karlsson. *A Higher Performance Or-Parallel Prolog System.* PhD thesis, Royal Institute of Technology, Stockholm, 1992.

[14] J. Riely and M. Hennessy. Distributed processes and location failures. In *Proc. ICALP'97*, pages 471–482 LNCS 1256, 1997.