



Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.W. de Bakker, J.N. Kok

Uniform abstraction, atomicity and contractions in
the comparative semantics of Concurrent Prolog

Computer Science/Department of Software Technology

Report CS-R8834

August

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

69 D 13, 69 D 41, 69 F 12, 69 F 32

Copyright © Stichting Mathematisch Centrum, Amsterdam

Uniform Abstraction, Atomicity and Contractions in the Comparative Semantics of Concurrent Prolog *

J.W. de Bakker

J.N. Kok[†]

Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

This paper shows the equivalence of two semantics for a version of Concurrent Prolog with non-flat guards: an operational semantics based on a transition system and a denotational semantics which is a metric semantics (the domains are metric spaces). We do this in the following manner: First a uniform language \mathcal{L} is considered, that is a language where the atomic actions have arbitrary interpretations. For this language we prove that a denotational semantics is correct with respect to the operational semantics. This result relies on Banach's fixed point theorem. Techniques stemming from imperative languages are used. Then we show how to translate a Concurrent Prolog program to a program in \mathcal{L} by selecting certain basic sets for \mathcal{L} and then instantiating the interpretation function for the atomic actions. In this way we induce the two semantics for Concurrent Prolog and the equivalence between the two semantics.

Remark: an extended abstract of this report will appear in Proc. Fifth Generation Computer Systems 1988 (FGCS '88), December 1988, Tokyo, Japan.

Key words and phrases: operational semantics, denotational semantics, parallelism, concurrent logic languages, semantic equivalence, metric topology.

1985 Mathematics Subject Classification: 68Q55, 68Q10.

1987 Computing Reviews Categories: D.1.3, D.3.1, F.1.2, F.3.2.

1 Introduction

'Pure' logic programming (LP) has by now a well-established semantic theory, described in, e.g. [L], [Ap] or [AvE]. Traditionally, at least three varieties of semantics are distinguished, viz. the 'declarative', 'procedural' and 'process' interpretations, and, for pure LP, it is a standard result that these semantics all coincide. For logic *programming languages* – with the emphasis now on programming *language* rather than on the underlying mathematical framework of pure LP – the situation is much less clear. Already for PROLOG, the prime example of a *sequential* language with its prescribed execution order (left-first selection and depth-first searching) and cut operator, the development of models situated in the tradition of programming language semantics – viz. *operational* and *denotational* –, and the establishment of the relationships between these models is a topic of recent and current research (e.g. [JM], [DM], [ABe], [Vi]).

*This work was carried out in the context of ESPRIT 415: Parallel Architectures and Languages for Advanced Information Processing – a VLSI-directed approach and in the context of LPC: the dutch National Concurrency Project, supported by the Netherlands Organization for Scientific Research (N.W.O.), grant 125-20-04

[†] Author's present address: Department of Computer Science, University of Utrecht, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

Next we consider the field of parallel logic languages. A forerunner was Van Emden and De Lucena's Predicate logic as a language for parallel logic programming [vEdL]. The most well-known parallel logic languages are Concurrent Prolog [Sh1], PARLOG [CG], [Ri] and Guarded Horn Clauses [Ue]. Further offsprings include (see [Ri] for more details and comparative language design comments) Flat Concurrent Prolog [MTSLS] and Flat Guarded Horn Clauses (discussed in [Ri]). Finally, we mention TFCP (Theoretical Flat Concurrent Prolog), described in [Sh2].

Parallelism in LP languages brings along the well-known (from the field of imperative languages) phenomena such as synchronization, suspension and deadlock, sending and receiving of messages, and process creation. Accordingly, it may be more advantageous to address the semantic issues in parallel LP following the tradition in imperative languages (emphasizing 'control') rather than that of pure LP (emphasizing 'logic').

For operational semantics the method of Structured Operational Semantics ([PI]) has become the standard tool. Systems of (possibly labeled) *transitions* are embedded into syntax directed deductive systems, providing a concise, powerful and flexible tool, as demonstrated by numerous applications (for parallel logic languages we mention [Sa]). For denotational semantics, a classification can be based on the underlying mathematical structures, thus leading to (at least) order-theoretic, metric, algebraic and category-theoretic models. We use metric structures as our main tool. The motivation for this is, briefly, the following: In a setting with parallelism, some form of 'history' of the computation (be it (sets of) sequences or traces, trees etc.) always plays a key role. Now, firstly, histories allow a natural metric (the longer the histories remain the same, the smaller their distance). Secondly, with respect to this metric many functions which play a role in our semantic domains are *contracting*. Contracting functions have *unique fixed points*, a fortunate circumstance which facilitates definitions of (the meaning of) recursion and of semantic operators, and which leads to an uniform and powerful technique in comparing concurrency semantics ([KR], [BM]).

A well-known phenomenon from imperative concurrency is that of *deadlock* (in LP returning as *failure*), inducing the need for a model which embodies more structure than just (sets of) sequences. A large variety of such 'branching time' models has by now been proposed, including ready sets, failure sets, and (synchronization) trees (see [OH] for a comparison). In case programming notions requiring branching time are combined with state transformations, the need for Plotkin's *resumptions* arises. We have developed our own (metric) way of solving domain equations which are at the bottom of such resumptions (described in [BZ] or [AR]). The introduction of committed-choice in parallel logic languages is a cause of deadlock (see for example [FL] for an analysis of this phenomenon).

In [K] we developed a denotational semantics for a version of Concurrent Prolog, employing the metric techniques (domains of processes in the resumptions style, contracting functions etc.) of [BZ] and successors. The branching structure built up as result of a computation before a *commit* is encountered, is collapsed, at the moment of such an encounter, into a set of streams. The paper [GCLS] develops, for the language TFCP [Sh2], operational and denotational semantics, the latter based on failure sets. Moreover, a fully abstractness theorem relating the two is presented. The third investigation we mention follows the approach of declarative semantics. In [LP2], a comprehensive analysis is provided of a number of synchronization mechanisms in parallel logic languages. This is achieved by defining a 'universal' language which incorporates all the features required to model the various synchronization mechanisms, and which contains as proper subsets Concurrent Prolog, Flat CP, GHC (and, a fortiori, Pure Horn Clause Logic). This language is given the usual semantic definitions on an extended Herbrand Universe, and all the standard results are shown to hold. The paper [LP2] is an extension of [LP1], dealing with the declarative semantics of logical read-only variables. Recently also a declarative semantics for Flat Guarded Horn Clauses was proposed [Le].

In this paper we develop an operational and a denotational semantics for a language \mathcal{L} . This language is *uniform* in the sense that some basic sets can have arbitrary interpretations. Another feature of \mathcal{L} is that we have an operator that turns its argument (any, possibly complex, statement s) into an elementary action or (control) *atom*, denoted by $[s]$. Hence our emphasis on *atomicity* in this investigation. We provide a proof of the correctness of the denotational semantics with respect to the operational semantics (we show that there exists a restriction operator which relates the

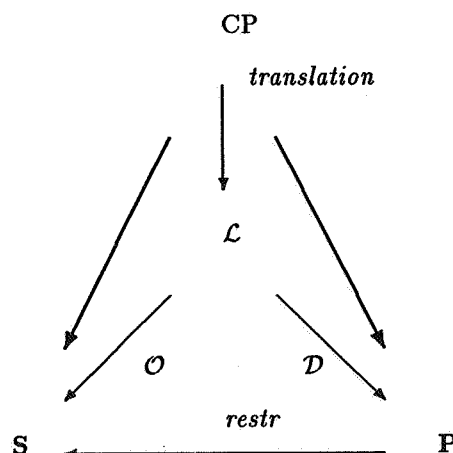


Figure 1: overview of the models

two). The operational semantics \mathcal{O} is based on a transition system. The denotational semantics \mathcal{D} is a metric semantics: the domains are metric spaces. A key role is played by *contractions*: they are used in almost all definitions. We have used *uniform abstraction*: In order to obtain the two semantics for Concurrent Prolog, we interpret the abstract sets of \mathcal{L} . For example, the set of elementary actions B will be the set of pairs (a_1, a_2) of (logical) atoms. The intended meaning of such a pair (a_1, a_2) of atoms is that we have to unify a_1 and a_2 . We then show how to translate a Concurrent Prolog program to a program in the uniform language \mathcal{L} . The denotational model that is induced in this way (from the denotational model for \mathcal{L}) resembles the model given in [K]. We also have an induced operational semantics and an induced relation between the two semantics. Figure 1 shows the relations. Note that the heavy lines in this figure refer to induced mappings only.

We think that the uniform abstraction procedure of first giving semantics to an uniform language and then the interpretation, gives more insight into the model. Moreover, we have the automatic link with an operational model.

The idea of a translation of Concurrent Prolog is already present in [Be]. In that paper a translation to Milner's CCS (Calculus of Communicating Systems) is provided. The recursion structure that is used in the paper is different: a clause is modeled by an agent which tries continuously to apply itself. In our model the equivalent of clauses is (recursive) procedures. The model in [Be] is based on synchronous communication, which is not present in our model.

We treat a larger subset of Concurrent Prolog than [GCLS]. The main difference is that we allow non-flat guards. This leads to more complex semantic domains: we have to introduce the notion of atomicity. One of the nice points of [GCLS] is that it makes clear what can be observed from a Concurrent Prolog program: for example that we can distinguish between failure and deadlock. They prove that their semantics is fully abstract with respect to the operational semantics. If we take the same observation criteria, we can adapt our semantic model (restricted to the subset considered by [GCLS]) in such a way that it is fully abstract along the lines of the methods described in [Ru]. A point of further research is whether or not non-flat guards influence these results. Following Apt and Plotkin ([AP]) we recall that in the case of unbounded nondeterminism (caused by non-flat guards) it might be impossible to assign a fully abstract semantics.

We give an outline of the rest of our paper. Metric topological preliminaries are given in section 2. Section 3 describes the language \mathcal{L} with its operational semantics \mathcal{O} and in section 4 the denotational semantics is defined. Section 5 gives the relationship between \mathcal{O} and \mathcal{D} . Finally,

section 6 provides the translation from Concurrent Prolog to \mathcal{L} . There are two appendices: the first one shows the compactness of a certain set (this result is used to show that one of the semantic models is welldefined) and in the second appendix we treat the extended unification of Concurrent Prolog.

2 Metric Preliminaries

We give in this section some basic definitions and properties about metric spaces. Let \mathbf{N} be the set of natural numbers. For further reference we suggest [En].

Definition 2.1 (Metric Spaces) A metric space is a pair (M, d) with M a non empty set and d a mapping $d : M \times M \rightarrow [0, 1]$ (a metric distance), which satisfies the following properties:

1. $\forall x, y \in M [d(x, y) = 0 \Leftrightarrow x = y]$,
2. $\forall x, y \in M [d(x, y) = d(y, x)]$,
3. $\forall x, y, z \in M [d(x, y) \leq d(x, z) + d(z, y)]$.

A metric space is called an ultrametric space if we replace 3 by the stronger 3':

$$3'. \forall x, y, z \in M [d(x, y) \leq \max(d(x, z), d(z, y))].$$

Definition 2.2 Let (M, d) be metric space. Let $(x_i)_i$ be a sequence in M .

1. We say that $(x_i)_i$ is a Cauchy sequence whenever we have

$$\forall \epsilon > 0 \exists N \in \mathbf{N} \forall n, m > N [d(x_n, x_m) < \epsilon].$$

2. Let $x \in M$. We say that $(x_i)_i$ converges to x and call x the limit of $(x_i)_i$ whenever we have

$$\forall \epsilon > 0 \exists N \in \mathbf{N} \forall m > N [d(x, x_m) < \epsilon]$$

Such a sequence we call convergent. Notation: $\lim_{i \rightarrow \infty} x_i = x$.

3. The metric space (M, d) is called complete whenever each Cauchy sequence converges to an element of M .

Definition 2.3 Let (M_1, d_1) , (M_2, d_2) be metric spaces. Let $0 \leq c < 1$. A function f from M_1 to M_2 which satisfies

$$\forall x, y \in M [d_2(f(x), f(y)) \leq c \cdot d_1(x, y)]$$

we call contracting.

Theorem 2.4 (Banach's fixed point theorem) Let (M, d) be a complete metric space and $f : M \rightarrow M$ a contracting function. Then there exists an $x \in M$ such that the following holds:

1. $f(x) = x$ (x is a fixed point of f),
2. $\forall y \in M [f(y) = y \Rightarrow y = x]$ (x is unique).

Definition 2.5 (closed subsets) A subset X of a metric space (M, d) is called closed whenever each Cauchy sequence of elements in X converges to an element of X .

Definition 2.6 The closure $Cl(X)$ of a subset X of a metric space is the set $\{\lim_{i \rightarrow \infty} y_i : \forall i [y_i \in X] \wedge (y_i)_i \text{ is a Cauchy sequence}\}$.

Definition 2.7 (compact subsets) A subset X of a metric space (M, d) is called compact whenever each sequence of elements in X has a converging subsequence.

Definition 2.8 Let (M, d) (M_1, d_1) (M_2, d_2) be metric spaces.

1. We define a metric d on the functions in $M_1 \rightarrow M_2$ as follows. For $f_1, f_2 \in M_1 \rightarrow M_2$

$$d(f_1, f_2) = \sup\{d_2(f_1(x), f_2(x)) : x \in M_1\}$$

2. Let $\mathcal{P}_{co}(M) = \{X \subset M : X \text{ is compact and non empty}\}$ and $\mathcal{P}_{ci}(M) = \{X \subset M : X \text{ is closed and non empty}\}$. We define a metric d_H on both $\mathcal{P}_{co}(M)$ and $\mathcal{P}_{ci}(M)$, called the Hausdorff distance, as follows. For every $X, Y \in \mathcal{P}_{co}(M)$ ($\in \mathcal{P}_{ci}(M)$)

$$d_H(X, Y) = \max\{\sup\{d(x, Y) : x \in X\}, \sup\{d(y, X) : y \in Y\}\}$$

where $d(x, Z) = \inf\{d(x, z) : z \in Z\}$ for every $Z \subset M$, $x \in M$.

Theorem 2.9 Let (M, d) , (M_1, d_1) , (M_2, d_2) be complete (ultra)metric spaces. We have that $M_1 \rightarrow M_2$, $\mathcal{P}_{co}(M)$ and $\mathcal{P}_{ci}(M)$ (with the metrics defined above) are complete (ultra)metric spaces.

In the sequel we sometimes suppress the definitions of metrics. We then assume that they are constructed in the standard ways as outlined above.

3 Syntax and operational semantics

Assume given a (possibly infinite) set of atomic actions B , with typical element b . Let $Proc$, with typical element P , be a set of procedure variables. These two basic sets are used in

Definition 3.1 We define the set of statements \mathcal{L} , with typical element s , by the following grammar:

$$s ::= b \mid P \mid s_1; s_2 \mid s_1 + s_2 \mid s_1 \parallel s_2 \mid [s_1]$$

A statement s is one of the following six forms:

- an elementary action b
- a procedure variable P
- the sequential composition $s_1; s_2$ of statements s_1 and s_2
- the nondeterministic choice $s_1 + s_2$
- the concurrent execution $s_1 \parallel s_2$, modeled by arbitrary interleaving
- the atomic version $[s]$ of s , modeled by interpreting s as an elementary action

Assume given a set of states Σ , with typical element σ . Let $Int = B \rightarrow \Sigma \rightarrow_{\text{partial}} \Sigma$ be the set of interpretations and let f be a typical element of Int . Given an elementary action b and an initial state σ , $f(b)(\sigma)$ (if it exists) is the state after the execution of b in state σ . The set of declarations $Decl$ (with typical element d) has as elements functions from $Proc \rightarrow \mathcal{L}_g$, where \mathcal{L}_g (the set of guarded statements) is defined in

Definition 3.2 We define the set of guarded statements \mathcal{L}_g , with typical element g , by the following syntax:

$$g ::= b \mid g; s \mid g_1 + g_2 \mid g_1 \parallel g_2 \mid [g_1]$$

Note that $\mathcal{L}_g \subset \mathcal{L}$. Intuitively, a statement s is guarded if all procedure variables are preceded by some statement. A program is a triple (f, d, s) , where s is a statement, $d \in Decl$ is a declaration for the procedure variables in s and f is an interpretation of the atomic actions. Let $Prog$ be the set of programs. In the sequel we sometimes suppress the declaration and interpretation parts of a program: instead of writing (f, d, s) we write just s . The operational semantics for \mathcal{L} is based on a transition relation in the style of [Pl]. A transition relation describes the steps we can take during a computation. We use a special symbol E , which stands for termination. A step can change the state and the (rest of the) program we have to execute.

Definition 3.3 *Let*

$$\rightarrow \subseteq (Prog \times \Sigma \times (Prog \cup \{E\}) \times \Sigma)$$

be the smallest relation satisfying (writing $(s, \sigma) \rightarrow (s', \sigma')$ for $(s, \sigma, s', \sigma') \in \rightarrow$ and $(s, \sigma) \rightarrow (E, \sigma')$ for $(s, \sigma, E, \sigma') \in \rightarrow$ and writing $A \rightarrow A_1 | \dots | A_n \Rightarrow B \rightarrow B_1 | \dots | B_n$ for $A \rightarrow A_1 \Rightarrow B \rightarrow B_1 \wedge \dots \wedge A \rightarrow A_n \Rightarrow B \rightarrow B_n$ where A, B are typical elements of $(Prog \cup \{E\}) \times \Sigma$)

- $(b, \sigma) \rightarrow (E, f(b)(\sigma))$ if $f(b)(\sigma)$ exists
- $(d(P), \sigma) \rightarrow (s, \sigma') | (E, \sigma') \Rightarrow (P, \sigma) \rightarrow (s, \sigma') | (E, \sigma')$
- $(s_1, \sigma_1) \rightarrow (s_2, \sigma_2) | (E, \sigma_2) \Rightarrow$

$$\begin{aligned} &(s_1; s, \sigma_1) \rightarrow (s_2; s, \sigma_2) | (s, \sigma_2) \\ &(s_1 \parallel s, \sigma_1) \rightarrow (s_2 \parallel s, \sigma_2) | (s, \sigma_2) \\ &(s \parallel s_1, \sigma_1) \rightarrow (s \parallel s_2, \sigma_2) | (s, \sigma_2) \\ &(s + s_1, \sigma_1) \rightarrow (s_2, \sigma_2) | (E, \sigma_2) \\ &(s_1 + s, \sigma_1) \rightarrow (s_2, \sigma_2) | (E, \sigma_2) \end{aligned}$$
- $(s, \sigma) \rightarrow^* (E, \sigma') \Rightarrow ([s], \sigma) \rightarrow (E, \sigma')$ (writing \rightarrow^* for the transitive closure of \rightarrow)

The last rule takes several transitions together: in order to get a step from $([s], \sigma)$ we analyse sequences of steps from (s, σ) . We have the following lemma:

Lemma 3.4 *For all $s \in \mathcal{L}$ and $\sigma \in \Sigma$, the set*

$$\{s' \in \mathcal{L} : \exists \sigma' \in \Sigma [(s, \sigma) \rightarrow (s', \sigma')]\}$$

is a finite set.

Please note that the set (for any $s \in \mathcal{L}$ and $\sigma \in \Sigma$) $\{(s', \sigma') \in \mathcal{L} \times \Sigma : (s, \sigma) \rightarrow (s', \sigma')\}$ is in general an infinite set.

We use the transition relation to give an operational semantics: we collect the sequence of states during a computation. Such a sequence can be finite or infinite. We also signal deadlock by a special symbol δ . Deadlock means that from a configuration (s, σ) no transition is possible. This can happen because Int contains partial functions. Let Σ^* (Σ^ω) denote the collection of all finite (infinite) words over Σ . Let x be a typical element of Σ^* and let y be a typical element of $\Sigma^\omega = \Sigma^* \cup \Sigma^\omega$. Let Σ^+ be Σ^* without the empty word and let $\Sigma_\delta^{st} = \Sigma^* \cdot \{\delta\} \cup \Sigma^+ \cup \Sigma^\omega$. Let z be a typical element of Σ_δ^{st} . Put $\mathbf{S} = \Sigma \rightarrow \mathcal{P}(\Sigma_\delta^{st})$: the set of functions from Σ to subsets of Σ_δ^{st} . The operational semantics is given in

Definition 3.5 *Let*

$$\mathcal{O} : Prog \rightarrow \mathbf{S}$$

be given by

$$\mathcal{O}(s) = \lambda \sigma. \begin{aligned} &\{\sigma_1 \dots \sigma_n : (s, \sigma) \rightarrow (s_1, \sigma_1) \dots \rightarrow (E, \sigma_n)\} \cup \\ &\{\sigma_1 \dots \sigma_n \dots : (s, \sigma) \rightarrow (s_1, \sigma_1) \dots \rightarrow (s_n, \sigma_n) \rightarrow \dots\} \cup \\ &\{\sigma_1 \dots \sigma_n \delta : (s, \sigma) \rightarrow (s_1, \sigma_1) \dots \rightarrow (s_n, \sigma_n) \not\rightarrow\} \end{aligned}$$

(writing $(s_n, \sigma_n) \not\rightarrow$ for $\forall s, \sigma [(s_n, \sigma_n, s, \sigma) \not\rightarrow \wedge (s_n, \sigma_n, E, \sigma) \not\rightarrow]$)

Note that the operational semantics is not compositional: take for example: $B = \{v := 1, v := 2, v := v + 1\}$, $\Sigma = \mathbb{N}$ (the set of integers), The state records the value of v . Let $f \in \text{Int}$ be such that $f(v := 1)(\sigma) = 1$, $f(v := 2)(\sigma) = 2$, $f(v := v + 1)(\sigma) = \sigma + 1$. Take $s_1 = v := 1; v := v + 1$ and $s_2 = v := 1; v := 2$. We have $\mathcal{O}(s_1) = \mathcal{O}(s_2) = \lambda\sigma.\{12\}$, but $\mathcal{O}(s_1 \parallel s_1) = \lambda\sigma.\{1212, 1123\}$ and $\mathcal{O}(s_2 \parallel s_2) = \lambda\sigma.\{1212, 1122\}$.

We turn Σ_δ^{st} into a complete metric space in

Definition 3.6

1. Define for each $z \in \Sigma_\delta^{st}$, $z[n]$ as the prefix of length n , if this exists, and $z[n] = z$, otherwise.
2. We define a metric d_{st} on Σ_δ^{st} by putting $d_{st}(z_1, z_2) = 2^{-N}$ if $z_1 \neq z_2$ where $N = \sup\{n : z_1[n] = z_2[n]\}$ and $d_{st}(z_1, z_2) = 0$ if $z_1 = z_2$.

We have the following lemma:

Lemma 3.7 For any $s \in \mathcal{L}$ and $\sigma \in \Sigma$, $\mathcal{O}(s)(\sigma)$ is a closed set.

Proof Take any $s \in \mathcal{L}$ and $\sigma \in \Sigma$. Suppose $(y_i)_i$ is a Cauchy sequence in $\mathcal{O}(s)(\sigma)$. We show that $\lim_{i \rightarrow \infty} y_i \in \mathcal{O}(s)(\sigma)$. We only consider the case that $\lim_{i \rightarrow \infty} y_i \in \Sigma^\omega$ (otherwise $\lim_{i \rightarrow \infty} y_i$ is constant from some moment on). Suppose that $\lim_{i \rightarrow \infty} y_i = \sigma_1 \sigma_2 \dots$. For all i we have that $y_i \in \mathcal{O}(s)(\sigma)$ and this implies that for any i we can pick sequences $(s_{ij}, \sigma_{ij})_j$ such that

$$(s, \sigma) \rightarrow (s_{i1}, \sigma_{i1}) \rightarrow (s_{i2}, \sigma_{i2}) \rightarrow \dots$$

and

$$y_i = \sigma_{i1} \sigma_{i2} \dots$$

Because $(y_i)_i$ is a Cauchy sequence, an infinite number of σ_{i1} equals σ_1 . By lemma 3.4, there is only a finite number of possibilities for s_{i1} . Hence there exists a $s_1 \in \mathcal{L}$ such that an infinite number of tuples (s_{i1}, σ_{i1}) equals (s_1, σ_1) . We can continue this construction and find a sequence of statements $(s_i)_i$ such that

$$(s, \sigma) \rightarrow (s_1, \sigma_1) \rightarrow (s_2, \sigma_2) \rightarrow \dots$$

Hence $y = \sigma_1 \sigma_2 \dots \in \mathcal{O}(s)(\sigma)$. □

Next we give an alternative definition for the operational semantics. Lemma 3.4 is used to show that this definition is well-defined. In the proof that the two definitions of \mathcal{O} coincide we use lemma 3.7.

From this moment on, we restrict \mathbf{S} to the set of functions from Σ to the closed subsets of Σ_δ^{st} : $\mathbf{S} = \Sigma \rightarrow \mathcal{P}_{cl}(\Sigma_\delta^{st})$. This enables us to assign a metric to \mathbf{S} in the standard way described in the previous section.

Definition 3.8 (alternative definition for \mathcal{O}) Let $\mathcal{O} : \text{Prog} \rightarrow \mathbf{S}$ be the unique fixed point of the contraction $\Delta : (\text{Prog} \rightarrow \mathbf{S}) \rightarrow (\text{Prog} \rightarrow \mathbf{S})$ which is defined as follows.

$$\Delta(F)(s) = \lambda\sigma. \{ \delta : (s, \sigma) \not\rightarrow \} \cup \{ \sigma_1 : (s, \sigma) \rightarrow (E, \sigma_1) \} \cup \bigcup \{ \sigma_1 \cdot F(s_1)(\sigma_1) : (s, \sigma) \rightarrow (s_1, \sigma_1) \}$$

First we show that the definition above is welldefined, i.e. that for any F, s, σ we have that $\Delta(F)(s)(\sigma)$ is a closed set. Take any F, s, σ and a Cauchy sequence $(z_i)_i$ in $\Delta(F)(s)(\sigma)$. There exists an infinite subsequence of $(z_{f(i)})_i$ ($f : \mathbb{N} \rightarrow \mathbb{N}$ monotonic) in one of the three following sets:

1. $\{ \delta : (s, \sigma) \not\rightarrow \}$

2. $\{\sigma_1 : (s, \sigma) \rightarrow (E, \sigma_1)\}$
3. $\bigcup\{\sigma_1 \cdot F(s_1)(\sigma_1) : (s, \sigma) \rightarrow (s_1, \sigma_1)\}$

We consider only the third case. By lemma 3.4 we know that there is only a finite number of possibilities for s_1 . Hence we can pick a σ_1 and a monotonic function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that $(z_{g(f(i))})_i$ is an infinite subsequence of $(z_{f(i)})_i$ in $\{\sigma_1 \cdot F(s_1)(\sigma_1) : (s, \sigma) \rightarrow (s_1, \sigma_1)\}$. (Only σ_1 is free.) Because $(z_{g(f(i))})_i$ is an infinite subsequence of a Cauchy sequence, we can find a σ_1 and a monotonic function $h : \mathbb{N} \rightarrow \mathbb{N}$ such that $(z_{h(g(f(i))))_i$ is an infinite sequence in $\{\sigma_1 \cdot F(s_1)(\sigma_1) : (s, \sigma) \rightarrow (s_1, \sigma_1)\}$. Because (by definition of F) $F(s_1)(\sigma_1)$ is a closed set we have that $\{\sigma_1 \cdot F(s_1)(\sigma_1) : (s, \sigma) \rightarrow (s_1, \sigma_1)\}$ is a closed set. Hence the infinite subsequence $(z_{h(g(f(i))))_i$ of $(z_i)_i$ converges to an element in $\Delta(F)(s)(\sigma)$. So also the whole Cauchy sequence converges to the same element in $\Delta(F)(s)(\sigma)$.

Next we show that the two definitions for \mathcal{O} coincide. By lemma 3.7 we know that $\mathcal{O} : Prog \rightarrow \mathbf{S}$ (\mathcal{O} of definition 3.5). It is not difficult to see that $\Delta(\mathcal{O}) = \mathcal{O}$. By Banach's fixed point theorem we have that the two fixed points are the same.

4 Denotational semantics

In this section we define a denotational semantics for \mathcal{L} . We call a semantics denotational if it is compositionally defined and tackles recursion with the help of fixed points. With each operator in \mathcal{L} we associate a semantic operator. The denotational semantics will be the fixed point of a higher-order operator. The denotational semantics will be based on domains which are metric spaces. These domains are defined as solutions of domain equations. The equations can be solved with techniques described in [AR] or [BZ].

In the construction of the domains for the denotational semantics we need an operator $\Sigma \square$ which is defined in

Definition 4.1 *Let (M, d) be a metric space. We define a metric \tilde{d} on $\Sigma \square M =_{def} \Sigma_\delta^{st} \cup \Sigma^+ \times M$ by putting*

- $\tilde{d}(z_1, z_2) = d_{st}(z_1, z_2)$ if $z_1, z_2 \in \Sigma_\delta^{st}$
- $\tilde{d}((z_1, m_1), (z_2, m_2)) = \begin{cases} d_{st}(z_1, z_2) & \text{if } z_1, z_2 \in \Sigma^+, m_1, m_2 \in M, z_1 \neq z_2 \\ 2^{-length(z_1)} \cdot d(m_1, m_2) & \text{if } z_1, z_2 \in \Sigma^+, m_1, m_2 \in M, z_1 = z_2 \end{cases}$
- $\tilde{d}(z_1, (z_2, m)) = \begin{cases} d_{st}(z_1, z_2) & \text{if } z_1 \in \Sigma_\delta^{st}, z_2 \in \Sigma^+, m \in M, z_1 \neq z_2 \\ 2^{-length(z_1)} & \text{if } z_1 \in \Sigma_\delta^{st}, z_2 \in \Sigma^+, m \in M, z_1 = z_2 \end{cases}$
- $\tilde{d}((z_1, m), z_2) = \begin{cases} d_{st}(z_1, z_2) & \text{if } z_2 \in \Sigma_\delta^{st}, z_1 \in \Sigma^+, m \in M, z_1 \neq z_2 \\ 2^{-length(z_1)} & \text{if } z_2 \in \Sigma_\delta^{st}, z_1 \in \Sigma^+, m \in M, z_1 = z_2 \end{cases}$

We have that $(\Sigma \square M, \tilde{d})$ is a complete ultrametric space if (M, d) is a complete ultrametric space. We briefly recall the notion of a (metric) domain equation. The general form of such an equation is $\mathbf{P} = F(\mathbf{P})$ or, more precisely, $(\mathbf{P}, d) \cong F((\mathbf{P}, d))$, where the mapping F maps metric spaces to metric spaces. Under certain conditions, we can find unique solutions (upto isometry) in the category of complete metric spaces. We have no room to discuss details. For our purposes, it is sufficient to know that $\mathbf{P} = \Sigma \rightarrow \mathcal{P}_{co}(\Sigma \square \mathbf{P})$ has a complete ultrametric space as solution. Formally, the metric on $\Sigma \rightarrow \mathcal{P}_{co}(\Sigma \square \mathbf{P})$ is derived from the metric on $M_1 = \mathbf{P}$, $M_2 = \Sigma \square M_1$, $M_3 = \mathcal{P}_{co}(M_2)$ and $M_4 = \Sigma \rightarrow M_3$.

Definition 4.2 *Let \mathbf{P} be the unique complete ultrametric space that satisfies*

$$\mathbf{P} = \Sigma \rightarrow \mathcal{P}_{co}(\Sigma \square \mathbf{P})$$

Elements of \mathbf{P} are called processes. Let p be a typical element of \mathbf{P} . Given an initial state σ , $p(\sigma)$ is a (compact) set. Elements of this set are either in Σ_δ^{st} or in $\Sigma^+ \times \mathbf{P}$. An element in Σ^+ (Σ^ω) can be seen as a (non) terminating computation, an element in $\Sigma^* \cdot \{\delta\}$ as a computation ending in deadlock. An element in $\Sigma^+ \times \mathbf{P}$ can be viewed as a terminating computation which has a resumption: after the computation (which is finite), it turns itself into another process.

For each syntactic operator in \mathcal{L} we define a semantic operator. The semantic operators corresponding with $;$, $+$ and \parallel will be of type $\mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}$ and the semantic operator corresponding to $[\cdot]$ will be of type $\mathbf{P} \rightarrow \mathbf{P}$.

Definition 4.3 *The operators \otimes , $\tilde{+}$, $\tilde{\parallel}$: $\mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}$ and the operator *stream*: $\mathbf{P} \rightarrow \mathbf{P}$ are defined as follows. Let*

$$p_1 \tilde{+} p_2 = \lambda \sigma. (p_1(\sigma) \cup p_2(\sigma))$$

and let $\otimes, \tilde{\parallel}$ be the unique fixed points of the contractions $\Phi_\otimes, \Phi_{\tilde{\parallel}}$: $(\mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}) \rightarrow (\mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P})$ that are defined as follows

$$\begin{aligned} \Phi_\otimes(F)(p_1, p_2) = \lambda \sigma. & \{z : z \in p_1(\sigma) \wedge z \in \Sigma^\omega \cup \Sigma^* \cdot \{\delta\}\} \cup \\ & \{(z, p_2) : z \in p_1(\sigma) \wedge z \in \Sigma^+\} \cup \\ & \{(z, F(p, p_2)) : (z, p) \in p_1(\sigma)\} \end{aligned}$$

$$\Phi_{\tilde{\parallel}}(F)(p_1, p_2) = \Phi_\otimes(F)(p_1, p_2) \tilde{+} \Phi_\otimes(F)(p_2, p_1)$$

and let *stream*: $\mathbf{P} \rightarrow \mathbf{P}$ be the unique fixed point of the contraction Φ_{stream} : $(\mathbf{P} \rightarrow \mathbf{P}) \rightarrow (\mathbf{P} \rightarrow \mathbf{P})$ that is defined by

$$\Phi_{stream}(F)(p) = \lambda \sigma. \{z \in \Sigma_\delta^{st} : z \in p(\sigma)\} \cup \{x\sigma'z : (x\sigma', p') \in p(\sigma) \wedge z \in F(p')(\sigma')\}$$

We give some explanation for the *stream* operation. First note that for all processes p and states σ we have that $stream(p)(\sigma) \subseteq \Sigma_\delta^{st}$. Hence we can say that the operator *stream* removes the tree-like structure of the process. It also removes the interleaving points of a process. Processes allow for interleaving: an element of $p(\sigma)$ can be of the form (x, p') . After computation x it turns itself into p' . Before starting the computations in p' , other processes can do some computations. Accordingly we say that between x and p' we have an interleaving point. The operator *stream* removes these points by passing the final state of the computation x as argument to the process p' .

In the sequel we use a left-merge operator

Definition 4.4 *Define $\tilde{\ll}$: $\mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}$ by $\tilde{\ll} = \Phi_\otimes(\tilde{\parallel})$.*

We often write $+$, \parallel , \ll rather than $\tilde{+}$, $\tilde{\parallel}$, $\tilde{\ll}$ if no confusion is possible. We have the following lemma.

Lemma 4.5 *For all $p_1, p_2, p'_1, p'_2 \in \mathbf{P}$ and for all $op \in \{\otimes, \parallel, \ll, +\}$ we have*

$$d(p_1 op p_2, p'_1 op p'_2) \leq \max\{d(p_1, p'_1), d(p_2, p'_2)\}$$

and

$$d(stream(p_1), stream(p_2)) \leq d(p_1, p_2)$$

A proof of a very similar lemma was given in [BZ]. Now we can define a denotational semantics for \mathcal{L} in

Definition 4.6 *Let $\mathcal{D} : \text{Prog} \rightarrow \mathbf{P}$ be the unique fixed point of the contraction $\Phi : (\text{Prog} \rightarrow \mathbf{P}) \rightarrow (\text{Prog} \rightarrow \mathbf{P})$ which is defined inductively as follows.*

- $\Phi(F)(b) = \lambda\sigma. \begin{cases} \{f(b)(\sigma)\} & \text{if } f(b)(\sigma) \text{ exists} \\ \{\delta\} & \text{otherwise} \end{cases}$
- $\Phi(F)(P) = \Phi(F)(d(P))$
- $\Phi(F)(s_1; s_2) = \Phi(F)(s_1) \otimes F(s_2)$
- $\Phi(F)(s_1 + s_2) = \Phi(F)(s_1) + \Phi(F)(s_2)$
- $\Phi(F)(s_1 \parallel s_2) = \Phi(F)(s_1) \parallel \Phi(F)(s_2)$
- $\Phi(F)([s]) = \text{stream}(\Phi(F)(s))$

5 Relation between the operational and denotational semantics

The operational semantics \mathcal{O} delivers linear-time objects (for a given state σ , $\mathcal{O}(s)(\sigma) \subseteq \Sigma_\delta^{st}$) whereas the denotational semantics \mathcal{D} delivers branching time objects in \mathbf{P} . We define a restriction operator *restr* which will link \mathcal{O} and \mathcal{D} : given a process p and an initial state σ , it delivers certain 'paths' in the process p . A path will be an element of Σ_δ^{st} . In the next definition we use the operator *last* which takes the last element of a word in Σ^+ .

Definition 5.1 *Let*

$$\text{restr} : \mathbf{P} \rightarrow \mathbf{S}$$

be the unique fixed point of the contraction

$$\Gamma : (\mathbf{P} \rightarrow \mathbf{S}) \rightarrow (\mathbf{P} \rightarrow \mathbf{S})$$

which is given by $\Gamma(F)(p)(\sigma) = \{\delta\}$ if $p(\sigma) \subset \Sigma^\omega \cup \Sigma^ \cdot \{\delta\}$ and*

$$\Gamma(F)(p)(\sigma) = \text{Cl} \left(\begin{array}{l} \{ \text{last}(x) : x \in \Sigma^+ \wedge x \in p(\sigma) \} \cup \\ \bigcup \{ \text{last}(x) \cdot F(p')(\text{last}(x)) : (x, p') \in \Sigma^+ \times \mathbf{P} \wedge (x, p') \in p(\sigma) \} \end{array} \right)$$

otherwise.

We have

Theorem 5.2 $\mathcal{O} = \text{restr} \circ \mathcal{D}$

In order to prove this theorem, we will define an intermediate semantics \mathcal{I} . It is called intermediate because it serves as an intermediate semantics between \mathcal{O} and \mathcal{D} : it is defined with the help of a transition system (like the operational semantics) and it delivers tree-like objects (like the denotational semantics). An essential further property of the intermediate semantics (compared to the operational semantics) is that it keeps the intermediate states in the computation. This facilitates the proof of the theorem which we prove in two steps. First we show that $\mathcal{I} = \mathcal{D}$ and secondly we show that $\mathcal{O} = \text{restr} \circ \mathcal{I}$.

We prove $\mathcal{I} = \mathcal{D}$ by showing that \mathcal{D} is a fixed point of the defining contraction of \mathcal{I} , and hence, by Banach's theorem, we have that $\mathcal{I} = \mathcal{D}$. We give the transition system for the intermediate semantics \mathcal{I} in

Definition 5.3 *Let*

$$\rightarrow \subseteq \text{Prog} \times \Sigma \times \Sigma^\infty \times (\text{Prog} \cup \{E\}) \times (\Sigma \cup \{\delta\})$$

be the smallest relation satisfying (writing $(s, \sigma) \xrightarrow{y} (s', \sigma')$ for $(s, \sigma, y, s', \sigma') \in \rightarrow$, $(s, \sigma) \xrightarrow{y} (E, \sigma')$ for $(s, \sigma, y, E, \sigma') \in \rightarrow$, $(s, \sigma) \xrightarrow{y} (s', \delta)$ for $(s, \sigma, y, s', \delta) \in \rightarrow$ and $(s, \sigma) \xrightarrow{y} (E, \delta)$ for $(s, \sigma, y, E, \delta) \in \rightarrow$ and writing $A \rightarrow A_1 | \dots | A_n \Rightarrow B \rightarrow B_1 | \dots | B_n$ for $A \rightarrow A_1 \Rightarrow B \rightarrow B_1 \wedge \dots \wedge A \rightarrow A_n \Rightarrow B \rightarrow B_n$ where A, B are typical elements of $(\text{Prog} \cup \{E\}) \times (\Sigma \cup \{\delta\})$)

- $(b, \sigma) \xrightarrow{\epsilon} (E, f(b)(\sigma))$ if $f(b)(\sigma)$ exists
- $(b, \sigma) \xrightarrow{\epsilon} (E, \delta)$ if $f(b)(\sigma)$ does not exist
- $(d(P), \sigma) \xrightarrow{y} (s, \sigma')|(E, \sigma')|(s, \delta)|(E, \delta) \Rightarrow (P, \sigma) \xrightarrow{y} (s, \sigma')|(E, \sigma')|(s, \delta)|(E, \delta)$
- $(s_1, \sigma_1) \xrightarrow{y} (s_2, \sigma_2)|(E, \sigma_2)|(s_2, \delta)|(E, \delta) \Rightarrow (s_1; s, \sigma_1) \xrightarrow{y} (s_2; s, \sigma_2)|(s, \sigma_2)|(s_2; s, \delta)|(s, \delta)$
 $(s_1 \parallel s, \sigma_1) \xrightarrow{y} (s_2 \parallel s, \sigma_2)|(s, \sigma_2)|(s_2 \parallel s, \delta)|(s, \delta)$
 $(s \parallel s_1, \sigma_1) \xrightarrow{y} (s \parallel s_2, \sigma_2)|(s, \sigma_2)|(s \parallel s_2, \delta)|(s, \delta)$
 $(s + s_1, \sigma_1) \xrightarrow{y} (s_2, \sigma_2)|(E, \sigma_2)|(s_2, \delta)|(E, \delta)$
 $(s_1 + s, \sigma_1) \xrightarrow{y} (s_2, \sigma_2)|(E, \sigma_2)|(s_2, \delta)|(E, \delta)$
- If $(s_1, \sigma_1) \xrightarrow{y_1} (s_2, \sigma_2) \xrightarrow{y_2} \dots (s_n, \sigma_n)$ then

$$(s_n, \sigma_n) \xrightarrow{y_n} (E, \sigma)|(E, \delta)|(s, \delta) \Rightarrow ([s_1], \sigma_1) \xrightarrow{y_1 \sigma_2 \dots \sigma_n y_n} (E, \sigma)|(E, \delta)|(s, \delta)$$

- $(s_1, \sigma_1) \xrightarrow{y_1} (s_2, \sigma_2) \xrightarrow{y_2} \dots (s_n, \sigma_n) \xrightarrow{y_n} \dots \Rightarrow ([s_1], \sigma_1) \xrightarrow{y_1 \sigma_2 \dots \sigma_n y_n \dots} (E, \delta)$

Note that we have defined two transition relations: one in definition 3.3 and the other in definition 5.3. The second relation is always written with a superscript. The following lemma holds

Lemma 5.4

$$\exists y[(s, \sigma) \xrightarrow{y} (s', \sigma')] \Leftrightarrow (s, \sigma) \rightarrow (s', \sigma')$$

$$\exists y[(s, \sigma) \xrightarrow{y} (E, \sigma')] \Leftrightarrow (s, \sigma) \rightarrow (E, \sigma')$$

It follows that

$$(s, \sigma) \not\rightarrow \Leftrightarrow \neg \exists y, s', \sigma' [(s, \sigma) \xrightarrow{y} (s', \sigma') \vee (s, \sigma) \xrightarrow{y} (E, \sigma')]$$

Next we give the intermediate semantics:

Definition 5.5 Let $\mathcal{I} : \text{Prog} \rightarrow \mathbf{P}$ be the unique fixed point of the contraction $\Psi : (\text{Prog} \rightarrow \mathbf{P}) \rightarrow (\text{Prog} \rightarrow \mathbf{P})$ which is defined as follows.

$$\begin{aligned} \Psi(F)(s) = \lambda \sigma. \quad & \{y\sigma' : (s, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\ & \{y\delta : (s, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\ & \{y\delta : (s, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\ & \{(y\sigma', F(s')) : (s, \sigma) \xrightarrow{y} (s', \sigma')\} \end{aligned}$$

In an appendix we show that Ψ is well-defined (i.e. for any F , s and σ we have that $\Psi(F)(s)(\sigma)$ is a compact set). We provide a lemma with properties of the defining contraction Ψ of \mathcal{I} :

Lemma 5.6

1. $\Psi(\mathcal{D})(b) = \mathcal{D}(b)$
2. $\Psi(\mathcal{D})(P) = \Psi(\mathcal{D})(d(P))$
3. $\Psi(\mathcal{D})(s_1; s_2) = \Psi(\mathcal{D})(s_1) \otimes \mathcal{D}(s_2)$
4. $\Psi(\mathcal{D})(s_1 + s_2) = \Psi(\mathcal{D})(s_1) + \Psi(\mathcal{D})(s_2)$
5. $\Psi(\mathcal{D})(s_1 \parallel s_2) = \Psi(\mathcal{D})(s_1) \parallel \mathcal{D}(s_2) + \Psi(\mathcal{D})(s_2) \parallel \mathcal{D}(s_1)$
6. $\Psi(\mathcal{D})([s]) \in \mathbf{P} \cap \mathbf{S}$

$$\begin{aligned}
7. \Psi(\mathcal{D})([s]) = \lambda\sigma. & \{y\sigma' : (s, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\
& \{y\delta : (s, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\
& \{y\delta : (s, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\
& \cup \{y\sigma' \cdot \Psi(\mathcal{D})([s'])(\sigma') : (s, \sigma) \xrightarrow{y} (s', \sigma')\}
\end{aligned}$$

We give some details of the proof of case 5.

$$\begin{aligned}
\Psi(\mathcal{D})(s_1 \parallel s_2) = \lambda\sigma. & \{y\sigma' : (s_1 \parallel s_2, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\
& \{y\delta : (s_1 \parallel s_2, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\
& \{y\delta : (s_1 \parallel s_2, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\
& \{(y\sigma', \mathcal{D}(s')) : (s_1 \parallel s_2, \sigma) \xrightarrow{y} (s', \sigma')\}
\end{aligned}$$

Because a transition from $(s_1 \parallel s_2, \sigma)$ never yields a configuration of the form (E, σ') or (E, δ) we have that

$$\begin{aligned}
\Psi(\mathcal{D})(s_1 \parallel s_2) = \lambda\sigma. & \{y\delta : (s_1 \parallel s_2, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\
& \{(y\sigma', \mathcal{D}(s')) : (s_1 \parallel s_2, \sigma) \xrightarrow{y} (s', \sigma')\}
\end{aligned}$$

By properties of the transition system we have

$$(s_1 \parallel s_2, \sigma) \xrightarrow{y} (s', \sigma')$$

if and only if

$$\exists s'_1 [(s_1, \sigma) \xrightarrow{y} (s'_1, \sigma') \wedge s' = s'_1 \parallel s_2]$$

or

$$\exists s'_2 [(s_2, \sigma) \xrightarrow{y} (s'_2, \sigma') \wedge s' = s_1 \parallel s'_2]$$

or

$$(s_1, \sigma) \xrightarrow{y} (E, \sigma') \wedge s' = s_2$$

or

$$(s_2, \sigma) \xrightarrow{y} (E, \sigma') \wedge s' = s_1.$$

By similar properties we have

$$(s_1 \parallel s_2, \sigma) \xrightarrow{y} (s', \delta)$$

if and only if

$$\exists s'_1 [(s_1, \sigma) \xrightarrow{y} (s'_1, \delta) \wedge s' = s'_1 \parallel s_2]$$

or

$$\exists s'_2 [(s_2, \sigma) \xrightarrow{y} (s'_2, \delta) \wedge s' = s_1 \parallel s'_2]$$

or

$$(s_1, \sigma) \xrightarrow{y} (E, \delta) \wedge s' = s_2$$

or

$$(s_2, \sigma) \xrightarrow{y} (E, \delta) \wedge s' = s_1.$$

Hence

$$\begin{aligned}
\Psi(\mathcal{D})(s_1 \parallel s_2) = \lambda\sigma. & \{y\delta : (s_1, \sigma) \xrightarrow{y} (s'_1, \delta)\} \cup \\
& \{y\delta : (s_2, \sigma) \xrightarrow{y} (s'_2, \delta)\} \cup \\
& \{y\delta : (s_1, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\
& \{y\delta : (s_2, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\
& \{(y\sigma', \mathcal{D}(s'_1 \parallel s_2)) : (s_1, \sigma) \xrightarrow{y} (s'_1, \sigma')\} \cup \\
& \{(y\sigma', \mathcal{D}(s_1 \parallel s'_2)) : (s_2, \sigma) \xrightarrow{y} (s'_2, \sigma')\} \cup \\
& \{(y\sigma', \mathcal{D}(s_2)) : (s_1, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\
& \{(y\sigma', \mathcal{D}(s_1)) : (s_2, \sigma) \xrightarrow{y} (E, \sigma')\}
\end{aligned}$$

Rearranging and using the compositionality of \mathcal{D} we obtain

$$\begin{aligned}
\Psi(\mathcal{D})(s_1 \parallel s_2) = \lambda\sigma. & \{y\delta : (s_1, \sigma) \xrightarrow{y} (s'_1, \delta)\} \cup \\
& \{y\delta : (s_1, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\
& \{(y\sigma', \mathcal{D}(s'_1) \parallel \mathcal{D}(s_2)) : (s_1, \sigma) \xrightarrow{y} (s'_1, \sigma')\} \cup \\
& \{(y\sigma', \mathcal{D}(s_2)) : (s_1, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\
& \{y\delta : (s_2, \sigma) \xrightarrow{y} (s'_2, \delta)\} \cup \\
& \{y\delta : (s_2, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\
& \{(y\sigma', \mathcal{D}(s_1) \parallel \mathcal{D}(s'_2)) : (s_2, \sigma) \xrightarrow{y} (s'_2, \sigma')\} \cup \\
& \{(y\sigma', \mathcal{D}(s_1)) : (s_2, \sigma) \xrightarrow{y} (E, \sigma')\}
\end{aligned}$$

and this equals

$$\Psi(\mathcal{D})(s_1 \parallel s_2) = \Psi(\mathcal{D})(s_1) \parallel \mathcal{D}(s_2) + \Psi(\mathcal{D})(s_2) \parallel \mathcal{D}(s_1)$$

Lemma 5.7 $\Psi(\mathcal{D}) = \mathcal{D}$

Proof We show that for all $s \in \mathcal{L}$

$$d(\Psi(\mathcal{D})(s), \mathcal{D}(s)) \leq \frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$$

This implies that $d(\Psi(\mathcal{D}), \mathcal{D}) \leq \frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$, i.e. $d(\Psi(\mathcal{D}), \mathcal{D}) = 0$, i.e. $\Psi(\mathcal{D}) = \mathcal{D}$.

We first prove it for $g \in \mathcal{L}_g$. We use structural induction on the elements of \mathcal{L}_g . We give only the cases $g; s, [g]$:

$$(g; s) \quad d(\Psi(\mathcal{D})(g; s), \mathcal{D}(g; s)) =$$

(lemma 5.6)

$$d(\Psi(\mathcal{D})(g) \otimes \mathcal{D}(s), \mathcal{D}(g) \otimes \mathcal{D}(s)) \leq$$

(lemma 4.5)

$$\max\{d(\Psi(\mathcal{D})(g), \mathcal{D}(g)), d(\mathcal{D}(s), \mathcal{D}(s))\} =$$

$$d(\Psi(\mathcal{D})(g), \mathcal{D}(g)) \leq$$

(induction)

$$\frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$$

$$([g]) \quad d(\Psi(\mathcal{D})([g]), \mathcal{D}([g])) \leq$$

(d is an ultrametric)

$$\max\{d(\Psi(\mathcal{D})([g]), \text{stream}(\Psi(\mathcal{D})(g))), d(\text{stream}(\Psi(\mathcal{D})(g)), \mathcal{D}([g]))\}$$

We show that

1. $d(\Psi(\mathcal{D})([g]), \text{stream}(\Psi(\mathcal{D})(g))) \leq \frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$
2. $d(\text{stream}(\Psi(\mathcal{D})(g)), \mathcal{D}([g])) \leq \frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$

1. $\Psi(\mathcal{D})([g]) =$
(lemma 5.6)

$$\begin{aligned} \lambda\sigma. & \{y\sigma' : (g, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\ & \bigcup \{y\sigma' \cdot \Psi(\mathcal{D})([s'])(\sigma') : (g, \sigma) \xrightarrow{y} (s', \sigma')\} \end{aligned}$$

On the other hand, we have

$$\text{stream}(\Psi(\mathcal{D})(g)) =$$

(definition of Ψ)

$$\begin{aligned} \text{stream}(\lambda\sigma. & \{y\sigma' : (g, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\ & \{(y\sigma', \mathcal{D}(s')) : (g, \sigma) \xrightarrow{y} (s', \sigma')\}) = \end{aligned}$$

(definition of *stream*)

$$\begin{aligned} \lambda\sigma. & \{y\sigma' : (g, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\ & \bigcup \{y\sigma' \cdot \text{stream}(\mathcal{D})(s')(\sigma') : (g, \sigma) \xrightarrow{y} (s', \sigma')\} = \end{aligned}$$

$$\begin{aligned} \lambda\sigma. & \{y\sigma' : (g, \sigma) \xrightarrow{y} (E, \sigma')\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (s', \delta)\} \cup \\ & \{y\delta : (g, \sigma) \xrightarrow{y} (E, \delta)\} \cup \\ & \bigcup \{y\sigma' \cdot \mathcal{D}([s']) : (g, \sigma) \xrightarrow{y} (s', \sigma')\} \end{aligned}$$

so

$$d(\Psi(\mathcal{D})([g]), \text{stream}(\Psi(\mathcal{D})(g))) \leq \frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$$

since $y\sigma'$ is not equal to the empty word (hence the factor $\frac{1}{2}$). Note that the last step does not use the induction hypothesis.

2. $d(\text{stream}(\Psi(\mathcal{D})(g)), \mathcal{D}([g])) =$
 $d(\text{stream}(\Psi(\mathcal{D})(g)), \text{stream}(\mathcal{D}(g))) \leq$

(lemma 4.5)

$$d(\Psi(\mathcal{D})(g), \mathcal{D}(g)) \leq$$

(induction hypothesis)

$$\frac{1}{2}.d(\Psi(\mathcal{D}), \mathcal{D})$$

Secondly, we extend \mathcal{L}_g to \mathcal{L} . We use structural induction on the elements of \mathcal{L} . All cases are the same as for \mathcal{L}_g , except for P (which is not present in the guarded case).

(P) By lemma 5.6 we have

$$\Psi(\mathcal{D})(P) = \Psi(\mathcal{D})(d(P))$$

and by the definition of \mathcal{D} we have

$$\mathcal{D}(P) = \mathcal{D}(d(P))$$

Hence

$$\begin{aligned} d(\Psi(\mathcal{D})(P), \mathcal{D}(P)) &= \\ d(\Psi(\mathcal{D})(d(P)), \mathcal{D}(d(P))) &\leq \\ (d(P) \text{ is a guarded statement}) & \\ \frac{1}{2} \cdot d(\Psi(\mathcal{D}), \mathcal{D}) & \end{aligned}$$

□

By Banach's fixed point theorem we have the following

Corollary 5.8 $\mathcal{I} = \mathcal{D}$

Lemma 5.9 $\mathcal{O} = \text{restr} \circ \mathcal{I}$

Proof We show that

$$\Delta(\text{restr} \circ \mathcal{I}) = \text{restr} \circ \mathcal{I}$$

where Δ is the defining contraction of \mathcal{O} . By the definition of Δ , $\Delta(\text{restr} \circ \mathcal{I})(s)(\sigma) = \{\delta\}$ if $(s, \sigma) \not\vdash$. Because $(s, \sigma) \not\vdash$ implies

$$\neg \exists y, s', \sigma' [(s, \sigma) \xrightarrow{y} (s', \sigma') \vee (s, \sigma) \xrightarrow{y} (E, \sigma')]$$

we have by definition of \mathcal{I} that

$$\mathcal{I}(s)(\sigma) \subset \Sigma^\omega \cup \Sigma^* \cdot \{\delta\}$$

i.e. $(\text{restr} \circ \mathcal{I})(s)(\sigma) = \{\delta\}$. Now assume that there are transitions possible from (s, σ) :

$$\begin{aligned} &\Delta(\text{restr} \circ \mathcal{I})(s)(\sigma) \\ = & \\ &\{\sigma' : (s, \sigma) \rightarrow (E, \sigma')\} \cup \\ &\cup \{\sigma' \cdot (\text{restr} \circ \mathcal{I})(s')(\sigma') : (s, \sigma) \rightarrow (s', \sigma')\} \\ = & \text{(because } \text{restr} \circ \mathcal{I} \in \text{Prog} \rightarrow \mathbf{S} \text{ we have that } \Delta(\text{restr} \circ \mathcal{I})(s)(\sigma) \text{ is a closed set by definition of } \Delta \text{ (cf. the justification for definition 3.8))} \\ &\text{Cl} \left(\begin{aligned} &\{\sigma' : (s, \sigma) \rightarrow (E, \sigma')\} \cup \\ &\cup \{\sigma' \cdot (\text{restr} \circ \mathcal{I})(s')(\sigma') : (s, \sigma) \rightarrow (s', \sigma')\} \end{aligned} \right) \\ = & \\ &\text{Cl} \left(\begin{aligned} &\{\text{last}(x \cdot \sigma') : (s, \sigma) \xrightarrow{x} (E, \sigma') \wedge x \in \Sigma^*\} \cup \\ &\cup \{\text{last}(x \cdot \sigma') \cdot (\text{restr} \circ \mathcal{I})(s')(\text{last}(x \cdot \sigma')) : (s, \sigma) \xrightarrow{x} (s', \sigma')\} \end{aligned} \right) \\ = & \\ &(\text{restr} \circ \mathcal{I})(s)(\sigma). \end{aligned}$$

□

6 Concurrent Prolog

In this section we apply the framework of the previous sections. We choose a set of elementary actions, a set of procedure variables, a set of states and an interpretation function in such a way that we obtain a denotational and an operational semantics for Concurrent Prolog. A Concurrent Prolog program is ‘translated’ to an element of *Prog*.

We first introduce the language Concurrent Prolog (CP for short) in an informal way. The reader not familiar with CP should consult [Sh1], the paper which introduces the concepts of CP.

Let a be a typical element of the set of atoms *Atom*. Atoms are built up in the usual way from constants, variables, functors and predicate symbols. In CP there is a special functor ? of arity one which is called the read-only functor. The paper [Sa] signals some problems with the interpretation of the read-only functor in [Sh1]. Saraswat gives in [Sa] an alternative interpretation for the read-only functor (input-only functor in his terms). This interpretation is an extension of normal unification. We take over his interpretation. In this section we do not give a formal definition of the extended unification. We provide an appendix in which we give the details. The rest of the paper can be read without knowledge of the exact details. We denote the extended unification function by $mgu_?$. It is a partial function on $Atom \times Atom$. If it is defined it delivers a substitution.

A CP program is a finite set of elements (called clauses) of the following form:

$$a \leftarrow a_1 \wedge \dots \wedge a_n | a_{n+1} \wedge \dots \wedge a_m$$

Both n, m can be 0. The bar | is called the commit operator, \bar{a} the head, $a_1 \wedge \dots \wedge a_n$ the guard and $a_{n+1} \wedge \dots \wedge a_m$ the body of the clause. If $n = 0$ we have an empty guard and if $n = m$ we have an empty body. Let *Clause* be the set of clauses and let c be a typical element of *Clause*. Besides a finite set of clauses, we also have a goal which is of the form

$$\bar{a}_1 \wedge \dots \wedge \bar{a}_k$$

If $k = 0$ we say that the goal is empty. The (interleaved) execution of a CP program goes as follows. We execute the goal given the identity substitution. The execution of a goal given a (current) substitution means that we try to resolve all the atoms in the goal until the goal is empty. If the goal is empty we return a substitution. In order to resolve an atom we unify it with the head of a clause (taking the current substitution into account) and we try to execute the guard (given the ‘new’ substitution resulting from the unification). The unification and the execution of the guard do not yet influence the current substitution. Only after the guard becomes empty we commit: we do not consider alternatives for this clause anymore and we replace the atom by the body of the clause and update the current substitution. The execution model described here is an interleaving model. We do not consider here a parallel model where we have truly parallel processes. In such a model we also would have to check if a substitution delivered by the execution of a guard matches with the current substitution.

We introduce disjoint sets of variables. This is done for technical reasons. During the process described above we replace atoms by bodies of clauses. In order to avoid clashes of variables, every time we rewrite an atom we ‘replace’ the variables in the clause by new ones. This is intuitively correct because clauses are assumed to be universally quantified. Therefore we partition the set of variables *Var* into infinite disjoint subsets Var_α , where α ranges over \mathbb{N}^* , the set of finite words of integers. Assume injections $\alpha : Var_\epsilon \rightarrow Var_\alpha$ (and their natural extensions to elements of *Atom*). Now we choose our basic sets: take Σ the set of substitutions, $B = Atom \times Atom$ and $Proc = Atom \times \mathbb{N}^*$ (where \mathbb{N}^* is the set of finite words of integers). A pair (a, α) in *Proc* specifies that we have to rewrite the atom a with a clause of the program in which the variables are taken from Var_α . Take

$$f(a_1, a_2)(\sigma) = mgu_?(a_1, \sigma(a_2)) \circ \sigma$$

if $mgu_?(a_1, \sigma(a_2))$ is defined and is undefined otherwise. The composition \circ is the usual composition of substitutions (see for example [Ap]).

Fix a CP program and a goal. We assume that all variables in the program and in the goal are taken from Var_ϵ . We define a function

$$stm : Clause \times Proc \rightarrow \mathcal{L}_g$$

by

$$\begin{aligned} stm(\bar{a} \leftarrow a_1 \wedge \dots \wedge a_n | a_{n+1} \wedge \dots \wedge a_m, (a, \alpha)) = \\ [(\alpha(\bar{a}), a); (\alpha(a_1), \alpha \cdot 1) \parallel \dots \parallel (\alpha(a_n), \alpha \cdot n)]; \\ (\alpha(a_{n+1}), \alpha \cdot n + 1) \parallel \dots \parallel (\alpha(a_m), \alpha \cdot m) \end{aligned}$$

Suppose the set of clauses is $\{c_1, \dots, c_k\}$. Define

$$(*) \quad d(P) = stm(c_1, P) + \dots + stm(c_k, P)$$

Assume the goal is $\bar{a}_1 \wedge \dots \wedge \bar{a}_k$. Take

$$s = (\bar{a}_1, 1) \parallel \dots \parallel (\bar{a}_k, k)$$

Some explanation is at its place. Execution of the goal consists in parallel execution of the k procedure variables $(\bar{a}_1, 1), \dots, (\bar{a}_k, k)$. When we call stm on a clause c and a pair (a, α) it considers what has to be done in order to rewrite atom a with clause c in which we have to take the variables from Var_α . Suppose $c = \bar{a} \leftarrow a_1 \wedge \dots \wedge a_n | a_{n+1} \wedge \dots \wedge a_m$. First we unify a with \bar{a} (the head of clause c). Because we have to take variables from Var_α we rename the variables in \bar{a} with the operator α : this results in the pair $(\alpha(\bar{a}), a)$. After this unification, we have to execute the guard of the clause c , i.e. $a_1 \wedge \dots \wedge a_n$. We can execute all the atoms (in which the variables are renamed by α) in parallel. In order to avoid clashes of variables, we specify that if $\alpha(a_i)$ is rewritten by a clause, variables in that clause are to be taken from $Var_{\alpha \cdot i}$. The resolving of the guard and the unification is not (yet) allowed to influence other computations. This is modeled by considering them to be an elementary action by placing $[\cdot]$ around the unification and the guard. After the execution of the guard, we continue with the execution of the body: again with the renaming and the specification of sets of variables.

Now we can also have a better understanding of the operator $restr$ (see definition 5.1). Alternative clauses are joined together by the $+$ operator in $(*)$. Semantically this means that they are alternative computations. As indicated above, the grainsize (the computations between two interleaving points) is the computation of guards. If there is a terminating guard computation, it can be chosen: deadlocking and/or nonterminating computations of guards need not to be considered anymore. If there are only deadlocking and/or nonterminating computations we never reach the commit and we have deadlock.

This translation induces an operational- and denotational semantics for Concurrent Prolog: We combine the translation to \mathcal{L} with the operational- and denotational semantics for \mathcal{L} . Also the equivalence (an operator linking the two semantics for Concurrent Prolog) is induced by the translation: we already have the restriction operator $restr$ that relates the two semantics for \mathcal{L} . This method of uniform abstraction gives in our opinion more insight into the semantic models than a direct definition would give. A direct definition yields a transition system in the style of [Sa] and a denotational semantics as in [K]. The proof of the equivalence between two such semantic definitions would be more difficult to understand (due to the interpretation of the abstract sets).

Note that the renaming of variables takes place at the level of syntax. An alternative would be to treat this renaming at a semantic level (for example in the states).

Acknowledgement: we acknowledge fruitful discussions on our work in the Amsterdam concurrency group, including Frank de Boer, Arie de Bruin, John-Jules Meijer, Jan Rutten and Erik de Vink. We thank Erik de Vink and Katuscia Palamidessi for the comments and suggestions made during the work.

References

- [AR] P. America, J.J.M.M. Rutten: *Solving reflexive domain equations in a category of complete metric spaces*, Proc. of the Third Workshop on Mathematical Foundations of Programming Language Semantics, Lecture Notes in Computer Science, Vol. 298, Springer (1988) 254-288.
- [Ap] K.R. Apt, *Introduction to logic programming*, Report CS-R8741, Centre for Mathematics and Computer Science, Amsterdam (1987), to appear as a chapter in Handbook of Theoretical Computer Science, North-Holland.
- [AvE] K.R. Apt, M.H. van Emden, *Contributions to the theory of logic programming*, JACM Vol. 29, No. 3, July 1982, pp. 841-862.
- [AP] K.R. Apt, G. Plotkin, *Countable nondeterminism and random assignment*, JACM Vol. 33, No. 4, October 1986, pp. 724-767.
- [ABe] B. Arbab, D.M. Berry, *Operational and denotational semantics of PROLOG*, Journal of Logic Programming 4 (1987) 309-330.
- [Be] L. Beckman, *Towards a formal semantics for concurrent logic programming languages*, Proc. of the Third International Conference on Logic Programming, Lecture Notes in Computer Science, Vol. 225, Springer (1986) 335-349.
- [B] J.W. de Bakker, *Comparative semantics for flow of control in logic programming without logic*, Report CS-R8840, Centre for Mathematics and Computer Science, Amsterdam (1988).
- [BM] J.W. de Bakker, J.-J.Ch. Meyer, *Metric semantics for concurrency*, Report CS-R8803, Centre for Mathematics and Computer Science, Amsterdam (1988) to appear in BIT.
- [BZ] J.W. de Bakker, J.I. Zucker, *Processes and the denotational semantics of concurrency*, Inform. and Control 54 (1982) 70-120.
- [BKPR] F.S. de Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten, *Control flow versus logic: a denotational and a declarative model for guarded horn clauses*, Report CS-R88.., Centre for Mathematics and Computer Science, Amsterdam (1988) to appear.
- [CG] K.L. Clark, S. Gregory, *Parallel programming in logic*, ACM Trans. Program. Lang. Syst. Vol. 8, 1 (1986) 1-49. Res. Report DOC 84/4, Dept. of Computing, Imperial College, London (1984).
- [DM] S.K. Debray, P. Mishra, *Denotational and operational semantics for PROLOG*, in Formal Description of Programming Concepts - III (M. Wirsing, ed.), North-Holland (1987) 245-269.
- [vEdL] M.H. van Emden, G.J. de Lucena, *Predicate logic as language for parallel programming*, Report CS79-15, Dept. of Computer Science, Univ. of Waterloo. Also in Logic Programming (K.L. Clark & S.A. Tarnlund, eds.) Academic Press, New York (1982) 189-198.
- [En] R. Engelking, *General Topology*, Polish Scientific Publishers 1977.
- [FL] M. Falaschi, G. Levi, *Operational and fixpoint semantics of a class of committed-choice languages*, Techn. Report, Dipartimento di Informatica, Universita di Pisa, Pisa (1988).
- [GCLS] R. Gerth, M. Codish, Y. Lichtenstein, E. Shapiro, *Fully abstract denotational semantics for concurrent prolog*, Proc. Logic In Computer Science (1988) 320-335.

- [HP] M. Hennessey, G.D. Plotkin, *Full abstraction for a simple parallel programming language*, in: Proceedings 8th MFCS (J. Becvar ed.), Lecture Notes in Computer Science, Vol. 74 Springer (1979) 108-120.
- [JM] N.D. Jones, A. Mycroft, *Stepwise development of operational and denotational semantics for prolog*, Proc. 1984 Int. Symp. on Logic Programming, Atlantic City, N.J. (1984).
- [K] J.N. Kok, *A compositional semantics for concurrent prolog*, Proc. Symp. on Theoretical Aspects Computer Science (R. Cori, M. Wirsing eds.), Lecture Notes in Computer Science Vol. 294, Springer (1988) 373-388.
- [KR] J.N. Kok, J.J.M.M. Rutten, *Contractions in comparing concurrency semantics*, Proc. 15th International Colloquium Automata, Languages and Programming (T. Lepistö, A. Salomaa eds.), Lecture Notes in Computer Science Vol. 317, Springer (1988) 317-332.
- [Le] G. Levi, *A new declarative semantics of flat guarded horn clauses*, Techn. Report, ICOT, Tokyo (1988).
- [LP1] G. Levi, C. Palamidessi, *The declarative semantics of logical read-only variables*, Proc. Symp. on Logic Programming, IEEE Comp. Society Press (1985) 128-137.
- [LP2] G. Levi, C. Palamidessi, *An approach to the declarative semantics of synchronization in logic languages*, Proc. 4th Int. Conference on Logic Programming, Melbourne, (1987) 877-893.
- [Ll] J.W. Lloyd, *Foundations of Logic Programming*, Springer (1984), (Second edition 1987).
- [MTSLS] C. Mierkowsky, S. Taylor, E. Shapiro, J. Levy, M. Safra, *The design and implementation of flat concurrent prolog*, Techn. Report CS85-09, Dept. of Applied Maths, Weizmann Institute, Israel (1985).
- [OH] E.-R. Olderog, C.A.R. Hoare, *Specification-oriented semantics for communicating processes*, in: Acta Informatica 23 (1986) 9-66.
- [Pl] G.D. Plotkin, *A structural approach to operational semantics*, Report DAIMI FN-19, Comp.Sci.Dept., Aarhus Univ. 1981.
- [Ri] G.A. Ringwood, *Parlog 86 and the dining logicians*, Comm. ACM, Vol. 31 (1988) 10-25.
- [Ru] J.J.M.M. Rutten, *Correctness and full abstraction of metric semantics for concurrency*, in Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds.), Lecture Notes in Computer Science, Springer, to appear.
- [Sa] V.A. Saraswat: *The concurrent logic programming language cp: definition and operational semantics*, in: Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages, Munich, West Germany, January 21-23, 1987, pp. 49-62.
- [Sh1] E.Y. Shapiro, *A subset of concurrent prolog and its interpreter*, Tech. Report TR-003, ICOT, Tokyo (1983).
- [Sh2] E.Y. Shapiro, *Concurrent prolog, a progress report*, in Fundamentals of Artificial Intelligence (W. Bibel, Ph. Jorrand, eds.), Lecture Notes in Computer Science, Vol 232, Springer (1987).
- [Ue] K. Ueda, *Guarded horn clauses*, Techn. Report TR-103, ICOT, Tokyo (1986).
- [Vi] E. de Vink, *Equivalence of an operational and a denotational semantics for a prolog-like language with cut*, Report IRR-151, Free University, Amsterdam (1988).

7 Appendix: welldefinedness of Ψ

In this appendix we show that the function Ψ as defined in definition 5.5 is welldefined: we show that $\Psi(F)(s)(\sigma)$ is a compact set for any F, s, σ . Before we state this as a lemma, we first define another transition system which enables us to analyse a computation $(s, \sigma) \xrightarrow{y} (s', \sigma')$ by giving intermediate statements and states. When we have established relations between this new transition system to the old one, we are able to use a standard method to show compactness.

We first extend the language \mathcal{L} to \mathcal{L}^{ext} by introducing the operators leftmerge \ll and rightmerge \gg :

$$s ::= b \mid P \mid s_1; s_2 \mid s_1 + s_2 \mid s_1 \parallel s_2 \mid [s_1] \mid s_1 \ll s_2 \mid s_1 \gg s_2$$

The intuition behind the left (right) merge is that before we do the normal merge, we first do a step from the left (right) statement. Next we give the transition relation

$$\rightarrow_3 \subset \mathcal{L}^{ext} \times \Sigma \times (\mathcal{L}^{ext} \cup \{E\}) \times (\Sigma \cup \{\delta\}).$$

It is the union of transition relations

$$\rightarrow_1, \rightarrow_2 \subset \mathcal{L}^{ext} \times \Sigma \times (\mathcal{L}^{ext} \cup \{E\}) \times (\Sigma \cup \{\delta\}).$$

The intuition behind \rightarrow_1 and \rightarrow_2 is that we do \rightarrow_2 transitions inside atomic brackets and that we do \rightarrow_1 transitions as long as we are outside the scope of such brackets. The transition relations $\rightarrow_1, \rightarrow_2$ are defined as follows:

Definition 7.1 *Let*

$$\rightarrow_1, \rightarrow_2 \subset \mathcal{L}^{ext} \times \Sigma \times (\mathcal{L}^{ext} \cup \{E\}) \times (\Sigma \cup \{\delta\})$$

be the smallest relation satisfying (writing $(s, \sigma) \rightarrow_i (s', \sigma')$ for $(s, \sigma, s', \sigma') \in \rightarrow_i$ and $(s, \sigma) \rightarrow_i (E, \sigma')$ for $(s, \sigma, E, \sigma') \in \rightarrow_i$ and $(s, \sigma) \rightarrow_i (E, \delta)$ for $(s, \sigma, E, \delta) \in \rightarrow_i$ and writing $A \rightarrow_i A_1 \mid \dots \mid A_n \Rightarrow B \rightarrow_i B_1 \mid \dots \mid B_n$ for $A \rightarrow_i A_1 \Rightarrow B \rightarrow_i B_1 \wedge \dots \wedge A \rightarrow_i A_n \Rightarrow B \rightarrow_i B_n$ where A, B are typical elements of $(\text{Prog} \cup \{E\}) \times \Sigma$) ($i = 1, 2$)

- $(b, \sigma) \rightarrow_1 (E, f(b)(\sigma))$ if $f(b)(\sigma)$ exists
- $(b, \sigma) \rightarrow_1 (E, \delta)$ otherwise
- $(d(P), \sigma) \rightarrow_i (s, \sigma') \mid (E, \sigma') \mid (E, \delta) \Rightarrow (P, \sigma) \rightarrow_i (s, \sigma') \mid (E, \sigma') \mid (E, \delta)$ ($i=1, 2$)
- $(s_1, \sigma_1) \rightarrow_1 (s_2, \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \Rightarrow$

$$\begin{aligned} &(s_1; s, \sigma_1) \rightarrow_1 (s_2; s, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s_1 + s, \sigma_1) \rightarrow_1 (s_2, \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \\ &(s + s_1, \sigma_1) \rightarrow_1 (s_2, \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \\ &(s \parallel s_1, \sigma_1) \rightarrow_1 (s \parallel s_2, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s_1 \parallel s, \sigma_1) \rightarrow_1 (s_2 \parallel s, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s_1 \ll s, \sigma_1) \rightarrow_1 (s_2 \parallel s, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s \ll s_1, \sigma_1) \rightarrow_1 (s \parallel s_2, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &([s_1], \sigma_1) \rightarrow_2 ([s_2], \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \end{aligned}$$
- $(s_1, \sigma_1) \rightarrow_2 (s_2, \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \Rightarrow$

$$\begin{aligned} &(s_1; s, \sigma_1) \rightarrow_2 (s_2; s, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s_1 + s, \sigma_1) \rightarrow_2 (s_2, \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \\ &(s + s_1, \sigma_1) \rightarrow_2 (s_2, \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \\ &(s_1 \parallel s, \sigma_1) \rightarrow_2 (s_2 \ll s, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s \parallel s_1, \sigma_1) \rightarrow_2 (s \ll s_2, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s_1 \ll s, \sigma_1) \rightarrow_2 (s_2 \ll s, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &(s \ll s_1, \sigma_1) \rightarrow_2 (s \ll s_2, \sigma_2) \mid (s, \sigma_2) \mid (E, \delta) \\ &([s_1], \sigma_1) \rightarrow_2 ([s_2], \sigma_2) \mid (E, \sigma_2) \mid (E, \delta) \end{aligned}$$

Note that transitions of the form $(s_1 \text{ op } s_2, \sigma) \rightarrow_1 \dots$ for $\text{op} \in \{\llbracket, \rrbracket\}$ are not needed if we consider only transition sequences from (s, σ) where $s \in \mathcal{L}$.

Lemma 7.2 For any $(s, \sigma) \in \mathcal{L}^{\text{ext}} \times \Sigma$ the sets

$$\{(s', \sigma') : (s, \sigma) \rightarrow_3 (s', \sigma')\}$$

and

$$\{(E, \sigma') : (s, \sigma) \rightarrow_3 (E, \sigma')\}$$

are finite.

Next we give a relationship between the transition relations:

Lemma 7.3 For all $n \geq 0$, $s, s', \sigma, \sigma', \sigma_1, \dots, \sigma_n, \dots$ we have:

1. $(s, \sigma) \xrightarrow{\sigma_1 \dots \sigma_n} (E, \sigma') \Rightarrow \exists s_1, \dots, s_n \in \mathcal{L}^{\text{ext}} [(s, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 \dots (s_n, \sigma_n) \rightarrow_3 (E, \sigma')]$
2. $(s, \sigma) \xrightarrow{\sigma_1 \dots \sigma_n} (s', \sigma') \Rightarrow \exists s_1, \dots, s_n \in \mathcal{L}^{\text{ext}} [(s, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 \dots (s_n, \sigma_n) \rightarrow_3 (s', \sigma')]$
3. $(s, \sigma) \xrightarrow{\sigma_1 \dots \sigma_n} (s', \delta) \Rightarrow \exists s_1, \dots, s_n \in \mathcal{L}^{\text{ext}} [(s, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 \dots (s_n, \sigma_n) \rightarrow_3 (E, \delta)]$
4. $(s, \sigma) \xrightarrow{\sigma_1 \dots \sigma_n} (E, \delta) \Rightarrow \exists s_1, \dots, s_n \in \mathcal{L}^{\text{ext}} [(s, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 \dots (s_n, \sigma_n) \rightarrow_3 (E, \delta)]$
5. $(s, \sigma) \xrightarrow{\sigma_1 \sigma_2 \dots} (E, \delta) \Rightarrow \exists s_1, s_2, \dots \in \mathcal{L}^{\text{ext}} [(s, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 (s_2, \sigma_2) \rightarrow_3 \dots]$
6. $(s, \sigma) \xrightarrow{\sigma_1 \sigma_2 \dots} (s', \delta) \Rightarrow \exists s_1, s_2, \dots \in \mathcal{L}^{\text{ext}} [(s, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 (s_2, \sigma_2) \rightarrow_3 \dots]$

The inverse implications are *not* true in general: take b_1, b_2 such that $f(b_1)(\sigma) = \sigma$ and $f(b_2)(\sigma)$ is undefined for all σ . We have

1. $(b_1; b_1, \sigma) \rightarrow_3 (b_1, \sigma) \rightarrow_3 (E, \sigma)$ but not $(b_1; b_1, \sigma) \xrightarrow{\sigma} (E, \sigma)$,
2. $(b_1; b_1; b_1, \sigma) \rightarrow_3 (b_1; b_1, \sigma) \rightarrow_3 (b_1, \sigma)$ but not $(b_1; b_1; b_1, \sigma) \xrightarrow{\sigma} (b_1, \sigma)$,
3. $(b_1; b_2, \sigma) \rightarrow_3 (b_2, \sigma) \rightarrow_3 (E, \delta)$ but not $(b_1; b_2, \sigma) \xrightarrow{\sigma} (s', \delta)$,
4. $(b_1; b_2, \sigma) \rightarrow_3 (b_2, \sigma) \rightarrow_3 (E, \delta)$ but not $(b_1; b_2, \sigma) \xrightarrow{\sigma} (E, \delta)$,
5. $d(P) = b_1; P$ and $(P, \sigma) \rightarrow_3 (P, \sigma) \rightarrow_3 (P, \sigma) \rightarrow_3 \dots$ but not $(P, \sigma) \xrightarrow{\sigma \sigma \dots} (E, \delta)$,
6. $d(P) = b_1; P$ and $(P, \sigma) \rightarrow_3 (P, \sigma) \rightarrow_3 (P, \sigma) \rightarrow_3 \dots$ but not $(P, \sigma) \xrightarrow{\sigma \sigma \dots} (s', \delta)$.

We can summarize these counterexamples above by saying that we do not have enough information about the grain size in the sequences of \rightarrow_3 transitions.

We introduce the notion of a substatement. A substatement intuitively is a part of a statement that can perform one step. The function *sub* delivers a set of substatements.

Definition 7.4

1. $\text{sub}(b) = \{b\}$
2. $\text{sub}(s_1; s_2) = \text{sub}(s_1)$
3. $\text{sub}(s_1 + s_2) = \text{sub}(s_1) \cup \text{sub}(s_2)$
4. $\text{sub}(s_1 \parallel s_2) = \text{sub}(s_1) \cup \text{sub}(s_2)$
5. $\text{sub}([s_1]) = \{[s_1]\}$
6. $\text{sub}(P) = \text{sub}(g)$ if $d(P) = g$.

With the notion of substatement we are able to state the following lemma:

Lemma 7.5 *For all $n \geq 0$, $s, \sigma, \sigma', \sigma_1, \dots, \sigma_n, \dots$ we have:*

1. $\exists s'[(s, \sigma) \xrightarrow{\sigma_1 \cdots \sigma_n} (s', \sigma')] \vee (s, \sigma) \xrightarrow{\sigma_1 \cdots \sigma_n} (E, \sigma')$
 \Leftrightarrow
 $\exists \tilde{s} \in \text{sub}(s) \exists s_1, \dots, s_n \in \mathcal{L}^{\text{ext}}[(\tilde{s}, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 \cdots (s_n, \sigma_n) \rightarrow_3 (E, \sigma')]$
2. $\exists s'[(s, \sigma) \xrightarrow{\sigma_1 \cdots \sigma_n} (s', \delta)] \vee (s, \sigma) \xrightarrow{\sigma_1 \cdots \sigma_n} (E, \delta)$
 \Leftrightarrow
 $\exists \tilde{s} \in \text{sub}(s) \exists s_1, \dots, s_n \in \mathcal{L}^{\text{ext}}[(\tilde{s}, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 \cdots (s_n, \sigma_n) \rightarrow_3 (E, \delta)]$
3. $\exists s'[(s, \sigma) \xrightarrow{\sigma_1 \sigma_2 \cdots} (s', \delta)] \vee (s, \sigma) \xrightarrow{\sigma_1 \sigma_2 \cdots} (E, \delta)$
 \Leftrightarrow
 $\exists \tilde{s} \in \text{sub}(s) \exists s_1, s_2, \dots \in \mathcal{L}^{\text{ext}}[(\tilde{s}, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 (s_2, \sigma_2) \rightarrow_3 \cdots]$

Now we turn to the welldefinedness of Ψ :

Lemma 7.6 *We have that $\Psi(F)(s)(\sigma)$ is a compact set for any F, s, σ .*

Proof

Pick arbitrary F, s, σ . Take an arbitrary infinite sequence in $\Psi(F)(s)(\sigma)$. There exists an infinite subsequence of this subsequence in one of the following four subsets of $\Psi(F)(s)(\sigma)$:

1. $\{y\sigma' : (s, \sigma) \xrightarrow{y} (E, \sigma')\}$
2. $\{y\delta : (s, \sigma) \xrightarrow{y} (E, \delta)\}$
3. $\{y\delta : (s, \sigma) \xrightarrow{y} (s', \delta)\}$
4. $\{(y\sigma', F(s')) : (s, \sigma) \xrightarrow{y} (s', \sigma')\}$

We only consider case 4. The other cases can be handled in a similar way. First note that there is only a finite number of statements s' such that

$$\exists \sigma' \exists y[(s, \sigma) \xrightarrow{y} (s', \sigma')]$$

Hence there exists an infinite subsequence of the form $(y_i \sigma'_i, F(s'))_i$ for some fixed s' . Hence it suffices to show that $(y_i \sigma'_i)_i$ has a converging subsequence.

For all i there exists a statement s'_i such that

$$(s, \sigma) \xrightarrow{y_i} (s'_i, \sigma'_i).$$

By lemma 7.5 we have that for each i there exist a substatement $\tilde{s}_i \in \text{sub}(s)$, integer $n_i \geq 0$, tuples $(s_{i1}, \sigma_{i1}), \dots, (s_{in_i}, \sigma_{in_i})$ such that

$$y_i = \sigma_{i1} \cdots \sigma_{in_i},$$

$$(\tilde{s}_i, \sigma) \rightarrow_3 (s_{i1}, \sigma_{i1}) \rightarrow_3 \cdots (s_{in_i}, \sigma_{in_i}) \rightarrow_3 (E, \sigma'_i).$$

There exists only a finite number of substatements of s . Hence an infinite number of the \tilde{s}_i is equal to a certain \tilde{s} . So we are able to pick an infinite subsequence $(y_{f(i)}\sigma'_{f(i)})_i$ of $(y_i\sigma'_i)_i$ where f is chosen such that f is monotonic and $\tilde{s}_{f(i)} = \tilde{s}$ for all i . For each $n \geq 1$ we can pick a tuple (s_n, σ_n) and a monotonic function $f_n : \mathbb{N} \rightarrow \mathbb{N}$ such that

1. if $n = 1$ then $(y_{(f_1 \circ f)(i)}\sigma'_{(f_1 \circ f)(i)})_i$ is an infinite subsequence of $(y_{f(i)}\sigma'_{f(i)})_i$
2. if $n > 1$ then $(y_{(f_{n+1} \circ \dots \circ f_1 \circ f)(i)}\sigma'_{(f_{n+1} \circ \dots \circ f_1 \circ f)(i)})_i$ is an infinite subsequence of $(y_{(f_n \circ \dots \circ f_1 \circ f)(i)}\sigma'_{(f_n \circ \dots \circ f_1 \circ f)(i)})_i$
3. $\forall i[(s_{(f_n \circ \dots \circ f_1 \circ f)(i)}, \sigma_{(f_n \circ \dots \circ f_1 \circ f)(i)}, n) = (s_n, \sigma_n)]$.

We have

$$(\tilde{s}, \sigma) \rightarrow_3 (s_1, \sigma_1) \rightarrow_3 (s_2, \sigma_2) \rightarrow_3 \dots$$

Take the infinite subsequence

$$(y_{(f_i \circ \dots \circ f_1)(i)}\sigma'_{(f_i \circ \dots \circ f_1)(i)})_i$$

of $(y_i\sigma'_i)_i$. Note that

$$\lim_{i \rightarrow \infty} y_{(f_i \circ \dots \circ f_1 \circ f)(i)}\sigma'_{(f_i \circ \dots \circ f_1 \circ f)(i)} = \sigma_1\sigma_2 \dots =^{def} y$$

and by lemma 7.5 either $\exists s'[(s, \sigma) \xrightarrow{y} (s', \delta)]$ or $(s, \sigma) \xrightarrow{y} (E, \delta)$. \square

8 Appendix: the function $mgu_?$

In this appendix we show a way to define the function $mgu_?$ based on the paper [Sa]. (Following [Sa] we interpret the read-only functor $?$ as an input-only functor.) There are other ways to define this function. This paper is independent of the way the $mgu_?$ function is defined.

Following Saraswat, we impose the following restrictions:

1. read-only functors are only allowed in heads of clauses,
2. if a subterm of a term is annotated, then also the term is annotated.

We formalize these points: Let t be a typical element of the set $Term$ of terms in which the read-only functor $?$ does not appear:

$$t ::= x \mid f(t_1, \dots, t_n) \quad (f \neq ? \wedge n \geq 0).$$

We extend $Term$ to $Term_?$ as follows. Let s be a typical element of the set $Term_?$:

$$s ::= t \mid ?(x) \mid ?(f(s_1, \dots, s_n)) \quad (f \neq ? \wedge n \geq 0).$$

Let $Atom$ contain elements of the form $P(t_1, \dots, t_n)$ and $Atom_?$ elements of the form $P(s_1, \dots, s_n)$ where P is a predicate symbol of arity n . The restrictions can now be stated as follows: elements of $Atom_?$ are only allowed as the heads of clauses. All other atoms (in goals, guards and bodies) are to be taken from $Atom$.

Assume that a variable in the head of a clause is annotated (preceeded by a read-only functor). The meaning of this annotation is that the variable should receive a (partial) value (i.e. a term that is not a variable) when it is unified with an atom in the goal.

We give an extended version of the unification algorithm based on [Ap] and [BKPR]: A finite subset of $Term_? \times Term_?$ we call *solved* if it is of the form

$$\{ \langle x_1, t_1 \rangle, \dots, \langle x_n, t_n \rangle \}$$

where for $1 \leq i \leq n$ we have that $x_i \in Var$, $t_i \in Term$, x_i does not occur in t_i and for $1 \leq i < j \leq n$ we have $x_i \neq x_j$. A finite solved subset of $Term_? \times Term_?$ determines a substitution θ as follows:

$$\theta(x_i) = t_i \quad (1 \leq i \leq n),$$

$$\theta(x) = x \quad (x \notin \{x_1, \dots, x_n\}).$$

Assume that $X, Y \subset Term? \times Term?$. We write $X \rightarrow Y$ if Y is obtained from X by choosing an element of one of the forms below and by performing the corresponding action.

1. $\langle x, x \rangle$: delete the pair,
2. $\langle ?(x), f(t_1, \dots, t_n) \rangle$: replace by $\langle x, f(t_1, \dots, t_n) \rangle$,
3. $\langle f(t_1, \dots, t_n), ?(x) \rangle$: replace by $\langle f(t_1, \dots, t_n), x \rangle$,
4. $\langle ?(f(s_1, \dots, s_n)), f(t_1, \dots, t_n) \rangle$: replace by $\langle s_1, t_1 \rangle, \dots, \langle s_n, t_n \rangle$,
5. $\langle f(t_1, \dots, t_n), ?(f(s_1, \dots, s_n)) \rangle$: replace by $\langle t_1, s_1 \rangle, \dots, \langle t_n, s_n \rangle$,
6. $\langle f(t_1, \dots, t_n), f(t'_1, \dots, t'_n) \rangle$: replace by $\langle t_1, t'_1 \rangle, \dots, \langle t_n, t'_n \rangle$,
7. $\langle t, x \rangle$: replace by $\langle x, t \rangle$,
8. $\langle x, t \rangle$ where x occurs in other pairs and x does not occur in t : apply substitution $x := t$ to all other pairs.

Now define

$$mgu? : Atom? \times Atom? \rightarrow Subst$$

by

$$mgu?(P(s_1, \dots, s_n), P(s'_1, \dots, s'_n)) = \theta$$

if there exist a $n \geq 0$ and X_1, X_2, \dots, X_n such that

$$\{\langle s_1, s'_1 \rangle, \dots, \langle s_n, s'_n \rangle\} \rightarrow X_1 \rightarrow X_2 \dots \rightarrow X_n$$

where X_n is in solved form and determines θ , and is undefined otherwise.