Science of Computer Programming 13 (1989/90) 237–264
North-Holland

COMPARATIVE SEMANTICS FOR PROLOG WITH CUT

E.P. de VINK

Department of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081, 1081 HV Amsterdam, Netherlands

Communicated by U. Montanari
Received November 1988
Revised October 1989

Abstract. An abstract language $\mathscr{B}$ embodying the flow of control component of PROLOG including the cut operator is considered. Both a denotational and operational model for $\mathscr{B}$ are developed and related by studying approximations of an intermediate semantics based on terminating transition systems. Interpretation and refinement of the models of $\mathscr{B}$ lead to a Plotkin-style operational semantics and an equivalent denotational semantics for PROLOG with cut.

1. Introduction

It is argued by several authors that the traditional fixpoint semantics for Horn clause logic, as described in, e.g., [1], is not suited for the logic programming language PROLOG (cf. [12, 19]). Especially the cut, added to Horn clause logic for efficiency reasons, affects the completeness of the refutation procedure (cf. [23]). Therefore, the standard declarative semantics using Herbrand models does not adequately capture the computational aspects of the PROLOG language. In the present paper we study the PROLOG-cut operator in a sequential environment augmented with backtracking. Our aim is to provide a denotational semantics that clarifies the flow of control aspects for the core of PROLOG with cut. Moreover, the semantics to be developed should be proved to be correct with respect to an operational model.

The question for equivalent denotational and operational models for PROLOG will be answered in two stages. First of all we separate the "logic" from the "control" (cf. [21]), and focus on the control flow aspects of PROLOG such as backtracking and the cut operator. Thus ignoring for a while the logic programming details, such as most general unifiers and renaming indices. We consider an abstract backtracking language $\mathscr{B}$ featuring such concepts as the cut operator, procedure calls and backtracking. One may call this "logic programming without logic" since we ignore any articulation in terms of resolution, [7]. Instead we consider arbitrary interpretations of actions as state transforming mappings. This leaves open the possibility for further specialization.

We develop both an operational and denotational semantics for $\mathscr{B}$. The operational semantics ($\mathscr{O}$) is based on the notion of a labeled transition system in the style of [11, 26]. The denotational semantics ($\mathscr{D}$) is similar to the denotational semantics

for PROLOG with cut of [19]. It uses the technique of an environment transformation. The meaning of a program $d|s$ is the result of evaluating its body s in an environment depending on the declaration d. Since we want our denotational semantics to be compositional, the cut operator forces the introduction of a special marker that is not needed in the operational case. This mere fact obstructs the equality of the operational and denotational model. Therefore, a transition-system-based intermediate semantics ($\mathscr{I}$) is defined that does not only resemble the operational semantics but also delivers this special cut marker in case a cut is encountered.

Equivalence of the operational and intermediate semantics modulo the cut marker is proved straightforward. Much more work is involved in comparing the intermediate and denotational model. Both the intermediate and denotational semantics are represented as least upper bounds of chains and we prove the equality of the approximating elements. (This technique, which first appeared in a preliminary version of this paper [15], is also used profitably in [13] in the setting of continuation semantics. In the context of complete metric spaces, it is implicit when using higher-order contractions with unique fixpoints for the semantical definition. See [10, 20].) We introduce complete partial orders of labeled transition systems, in which the notion of approximation is formalized. We consider so-called restricted intermediate systems that are induced by subsets of configurations with a bound on the nesting of procedure calls. By allowing a deeper nesting of calls we obtain a better approximation of the intermediate semantics. It will turn out that the kth intermediate approximation ($\mathscr{I}_k$) will correspond to the kth denotational one ($\mathscr{D}_k$).

Compositionality is a key property in the comparison of both kind of approximations. We establish compositionality of the $\mathscr{I}_k$ by induction on the *finite* length of the transition sequences involved. This termination property or Noetherianity is not shared by the intermediate semantics $\mathscr{I}$ (nor by the operational semantics $\mathscr{O}$). This makes a comparison with the denotational semantics on the level of approximations beneficial.

Having established equivalent denotational and operational models for the "control" component of PROLOG with cut, viz. for the abstract language $\mathscr{B}$, we return to our question for a semantics for PROLOG itself. The idea is to obtain such a semantics by refinement of the semantics for $\mathscr{B}$. We shall reformulate the concepts of $\mathscr{B}$ in the context of logic programming. Adding the "logic" to $\mathscr{B}$ amounts to consideration of a unification $t_1 = t_2$ as an action and of a conjunction of goals $G_1 \& G_2$ as a sequential composition. A state will be regarded as a substitution. Some care has to be taken with respect to procedure calls, i.e., atoms. For, in PROLOG an atom is replaced by the body of a clause provided that the atom and the head of the clause unify successfully. (Such a conditional rewriting is absent in the context of $\mathscr{B}$.) We shall use renaming indices to assure that the variables in the particular clause are fresh. Moreover, the effect of this unification has to be taken into account, since it updates the current substitution with a most general unifier. However, all this can be added quite painlessly to the already developed machinery. This is mainly due to flexibility of the $\mathscr{B}$-semantics with respect to the atomic actions. For we allow

in the setting of $\mathcal{B}$ *arbitrary* interpretations of them. In addition to the easy formulation of the semantical mappings it is straightforwardly checked that the de-uniformization process does not essentially affect the equivalence proof given for the operational and denotational models for $\mathcal{B}$. Therefore, correctness of the denotational semantics for PROLOG with cut with respect to the operational one is obtained in a clean way, without obscuring the proof by irrelevant details.

Related work on the denotational semantics for PROLOG with cut includes [3, 12, 13, 19]. In [3, 19] a direct Scott-Strachey-style denotational semantics is given, similar to the denotational semantics developed in the present paper. Their semantics are developed for the purpose of generating correct PROLOG interpreters and for proving termination properties, respectively. An operational semantics is also defined in [19]. However, no relationship between both models is established. Instead, confidence in the correctness of the denotational semantics is supplied by its systematic development. We extend upon their results by the formulation and proof of a correctness theorem which relates our (direct) denotational and operational models. The semantics of [12] mixes a direct and continuation style of denotational semantics. It uses sequences and cut flags on the one hand and declaration continuations on the other. However, it makes the semantical definition more complex and has its impact on the equivalence proof given. (In particular we doubt whether the induction hypothesis is applicable for Theorem 4.1, case 5 in [12, Appendix].) The continuation semantics in [13] uses continuations only, viz. success, failure and dump continuations. The equivalence proof for the denotational and operational models uses a technique of a hybrid semantics having both operational and denotational aspects. It is related to our approach in that it also makes comparisons at the level of approximations. Other work on PROLOG semantics from different perspectives and using several approaches includes [2, 4, 14, 16]. Finally, we mention [8] where a uniform language is used as a basis for a metric semantics of concurrent PROLOG.

The remainder of this paper is organized as follows. In Section 2 we give the definition of a labeled transition system and establish some general properties. Section 3 introduces the abstract language $\mathcal{B}$ and its operational semantics. The denotational semantics for $\mathcal{B}$ is the subject of Section 4. Having established, in Section 5, the equivalence of both models, we interpret, in Section 6, the abstract language in order to obtain an operational and denotational semantics for PROLOG with cut. Some final remarks are made in Section 7.

2. Labeled transition systems

In this section we introduce the notion of a labeled transition system. Collections of labeled transition systems are turned into a complete partial order (cpo) such that associating a valuation to a transition system becomes a continuous operation. In the realm of sequential programming we restrict ourselves to the deterministic case.

Definition 2.1. A *labeled (deterministic) transition system* T is an eight-tuple $\langle C, I, F, \Omega, \Lambda, D, \alpha, S\rangle$, where the set of configurations C is the disjoint union of I, F and $\{\Omega\}$, I is a set of internal configurations, F is a set of final configurations, Ω is the undefined configuration, Λ is the set of labels, the domain of values D is a cpo with a continuous composition or concatenation operator $\cdot$, the mapping $\alpha: F \cup \Lambda \to D$ is a valuation assigning a value to each final configuration and to each label, and S is a deterministic step or transition relation, i.e., a partial function $S: C \to_{\text{part}} \Lambda \times C$ with $dom(S) \subseteq I$.

Next we show how to extend the valuation α on final configurations to a valuation α_T on arbitrary configurations of a transition system T. Intuitively, α_T assigns to a configuration c the composition of the labels occurring at the arrows of the maximal transition sequence of the system T starting from c.

Definition 2.2. Let $T = \langle C, I, F, \Omega, \Lambda, D, \alpha, S\rangle$ be a labeled transition system. Let $\bot$ denote the bottom element of D. We associate with T a mapping $\alpha_T: C \to D$ defined as the least function in $C \to D$ such that

$$\alpha_T(c) = \begin{cases} \Omega, & \text{if } c = \Omega, \\ \alpha(c), & \text{if } c \in F, \\ \alpha(\lambda) \cdot \alpha_T(c'), & \text{if } (c, \lambda, c') \in S, \\ \bot, & \text{otherwise.} \end{cases}$$

Fix sets I and F of internal and final configurations, respectively. Fix an undefined configuration Ω, a set of labels Λ, a domain of values D, a valuation function $\alpha: F, \Lambda \to D$ and put $C = I \cup F \cup \{\Omega\}$. Let

$$TS = \{\langle C, I, F, \Omega, \Lambda, D, \alpha, S\rangle \mid S: C \to_{\text{part}} \Lambda \times C \text{ with } dom(S) \subseteq I\}$$

denote the collection of all labeled transition systems with configurations in C, internal configurations in I, final configurations in F, undefined configuration Ω, labels in Λ, domain of values D and valuation function α. In TS we identify a transition system with its transition relation. (In particular we may write $T(c)$ and $c \to_T^{\lambda} c'$ rather than $S(c)$ or $(c, \lambda, c') \in S$ for a transition system T with step relation S.)

We consider the set of configurations as a flat cpo with ordering $\leqslant_C$ and least element Ω. This induces an ordering $\leqslant_{\Lambda,C}$ on $\Lambda \times C$ and an ordering $\leqslant_{TS}$ on TS as follows:

$$(\lambda, c) \leqslant_{\Lambda,C} (\lambda', c') \Leftrightarrow (\lambda = \lambda') \,\&\, (c \leqslant_C c')$$

and

$$T \leqslant_{TS} T' \Leftrightarrow (dom(T) \subseteq dom(T')) \,\&\, (\forall c \in dom(T): T(c) \leqslant_{\Lambda,C} T'(c)).$$

So for labeled transition systems T and T' we have $T \leqslant_{TS} T'$ iff for each internal configuration c that admits a transition $c \to_T^{\lambda} c'$ in T, c admits a transition $c \to_{T'}^{\lambda} c''$

in T' such that these transitions have the same label and such that c' is the undefined configuration Ω or c' and c'' are equal.

We have that TS is a cpo when ordered by $\leqslant_{TS}$. The nowhere defined transition system $\emptyset$ is the least element of TS; for a chain $\langle T_k\rangle_k$ in TS the transition system T with

$$dom(T) = \bigcup_k dom(T_k) \quad \text{and} \quad T(c) = lub_k(T_k(c)),$$

for an appropriate chain in $\Lambda \times C$, acts as least upper bound. Moreover, the operation $\lambda T.\alpha_T : TS \to C \to D$ that assigns to a transition system the valuation it induces is continuous with respect to $\leqslant_{TS}$. (See [15] for more details.)

Remark. Let $I_0 \subseteq I_1 \subseteq \cdots$ be an infinite sequence of subsets of internal configurations such that $I = \bigcup_k I_k$. Put $C_k = I_k \cup F \cup \{\Omega\}$. Then we can construct for each $T \in TS$ a chain of approximations $\langle T_k\rangle_k$ of T in TS, where T_k is defined as the smallest labeled transition system such that

$$(c, \lambda, c') \in T_k \quad \text{if } ((c, \lambda, c') \in T) \,\&\, (c \in I_k) \,\&\, (c' \in C_k),$$

and

$$(c, \lambda, \Omega) \in T_k \quad \text{if } (c \in I_k) \,\&\, (\exists \bar{c} \in C \setminus C_k : (c, \lambda, \bar{c}) \in T).$$

Then it follows from the above that $T = lub_k(T_k)$ in TS. Moreover, for the valuations α and α_k of T and T_k, respectively, it holds that $\forall c \in C: \alpha(c) = lub_k(\alpha_k(c))$ with respect to the ordering of D.

T_k is called the restriction of T to I_k since only configurations in I_k act as a left-hand side. Note also that only configurations in C_k act as a right-hand side. We shall use this observation concerning restrictions in the congruence proof of the operational and denotational semantics.

In the sequel we shall use the cpos Σ^{st} and $\Sigma_\delta^{\mathrm{st}}$ as domain of values. (Cf. [5, 11, 24].) The collection Σ^{st} of streams over Σ and the collection $\Sigma_\delta^{\mathrm{st}}$ of δ-streams over Σ are defined by

$$\Sigma^{\mathrm{st}} = \Sigma^* \cup \Sigma^*.\bot \cup \Sigma^\omega,$$

and

$$\Sigma_\delta^{\mathrm{st}} = \Sigma^* \cup \Sigma^*.\bot \cup \Sigma^*.\delta \cup \Sigma^\omega,$$

respectively. The stream ordering $\leqslant_{\mathrm{st}}$ on Σ^{st} and $\Sigma_\delta^{\mathrm{st}}$ is defined by

$$x <_{\mathrm{st}} y \Leftrightarrow (\exists x' \in \Sigma^* \; \exists y' \in \Sigma_\delta^{\mathrm{st}} \setminus \{\bot\} : (x = x'\bot) \,\&\, (y = x'y')).$$

Intuitively, a stream x is (stream)less than a stream y exactly when x is unfinished, i.e., ending in $\bot$ and x can be extended to y by expanding the trailing bottom. (In [19] these domains are intuitively formulated in terms of equivalence classes of tail-lazy lists ordered by the prefix ordering.) The composition operator $\cdot : D \times D \to D$

is defined as the least mapping $\star$ such that $\varepsilon \star y = y$, $\perp \star y = \perp$ and $\sigma x' \star y = \sigma(x' \star y)$ for $D = \Sigma^{st}$ and additionally $\delta \star y = \delta$ for $D = \Sigma^{st}_{\delta}$. In particular we have

$$x \cdot y = x \quad \text{if } x \in \Sigma^*.\perp \cup \Sigma^*.\delta \cup \Sigma^{\omega}.$$

One can use the techniques described in [25] to justify such a definition and to establish the continuity of the defined operator.

3. Operational semantics of $\mathcal{B}$

In this section we introduce the abstract backtracking language $\mathcal{B}$ and define an operational semantics for it. $\mathcal{B}$ can be regarded as a uniform version—uniform in the sense of [9]—of PROLOG with cut, i.e., $\mathcal{B}$ reflects the control flow component of PROLOG. However, the statements in $\mathcal{B}$ are schematic in nature. All the logic present in PROLOG is abstracted away. The operational semantics of $\mathcal{B}$, which will be refined to a semantics for PROLOG with cut in Section 6, is based on the notion of a labeled transition system. For a program $d \mid s$ in $\mathcal{B}$, the declaration d will induce a transition system $\rightarrow_d$ while the statement s induces (given a state) an initial configuration. The operational semantics, then, is the sequence of labels of the maximal transition sequence with respect to $\rightarrow_d$ starting from the initial configuration with respect to s.

Definition 3.1. Fix a set of actions *Action* and a set of procedure names *Proc*. We define the set of elementary statements

$$EStat = \{a, \textbf{\textit{fail}}, !, s_1 \textbf{ \textit{or} } s_2, x \mid a \in Action, s_i \in Stat, x \in Proc\},$$

the set of statements

$$Stat = \{e_1 ; .. ; e_r \mid r \in \mathbb{N}, e_i \in EStat\}$$

and the set of declarations

$$Decl = \{x_1 \leftarrow s_1 : .. : x_r \leftarrow s_r \mid r \in \mathbb{N}, x_i \in Proc, s_i \in Stat, i \neq j \Rightarrow x_i \neq x_j\}.$$

The backtracking language $\mathcal{B}$ is defined by $\mathcal{B} = \{d \mid s \mid d \in Decl, s \in Stat\}$.

So an elementary statement is either an action in *Action*, the failure statement ***fail***, a PROLOG-like cut !, an alternative composition s_1 ***or*** s_2 or a procedure call x. A statement is a (possibly empty) sequential composition of elementary statements. The empty statement is denoted by ε. A declaration is a list of procedure definitions for different procedures names. Programs are made up from a declaration and a program body, i.e., a statement.

We let a range over *Action*, x over *Proc*, e over *EStat*, s over *Stat* and d over *Decl*. We write $x \leftarrow s \in d$ if $x \leftarrow s = x_i \leftarrow s_i$ (for some i) or if $s = \textbf{\textit{fail}}$ otherwise.

Example. From [13] we take the following example. Consider the context-free language $\mathscr{L}$ generated by the grammar

$$X \to YZ, \qquad Y \to aYa|bYb|a, \quad Z \to cZ|c.$$

$\mathscr{L}$ consists of palindromes over $\{a, b\}$ with a in the middle, followed by an arbitrary but positive number of c's. A parser for $\mathscr{L}$ is implemented by the $\mathscr{B}$-program $d|s$ where

$$d = x \leftarrow y;z : y \leftarrow a;y;a \textbf{ or } (b;y;b \textbf{ or } a) : z \leftarrow c;z \textbf{ or } c$$

and

$$s = x;\textbf{eoi}.$$

Intuitively, the actions a, b and c succeed if the corresponding symbol is currently read. Otherwise the actions fail and cause a backtrack to possible (stacked) alternatives with their own local states (i.e., their own tape head positions). Analogously, the action ***eoi*** succeeds if the whole input is scanned and fails otherwise. It is clear that once the palindrome part is recognized the alternative rules concerning the nonterminal Y do not have to be stacked any more. Here we can speed up rejections if we map $X \to YZ$ on $x \leftarrow y;!;z$ rather than on $x \leftarrow y;z$. We return to this example later.

So the cut ! provides a mechanism to discard of alternatives dynamically. To be more precise, execution of the cut amounts to commitment to the choices made since the statement in which the cut occurs is invoked. Consider for example $(a \textbf{ or } b);! \textbf{ or } c$. If a can be executed successfully in some state both (the occurrences of) the alternatives b and c are ignored in the rest of the computation. On the other hand, consider the declaration $x \leftarrow (a \textbf{ or } b);!$ and the statement $x \textbf{ or } c$. If a can be executed successfully the alternative b—as before—is discarded. However, the alternative c remains open since it is not within the scope of the cut in the body $(a \textbf{ or } b);!$ of the procedure x.

Let $d \in \mathit{Decl}$. The internal configurations of the transition system $\to_d$ associated with d are non-empty stacks. Each frame on a stack represents an alternative for the execution of some initial goal. As such a frame consists of a generalized statement and a local state taken from a set of states Σ. The state can be thought of as holding the values or bindings of the variables for a particular alternative. The generalized statement is composed from ordinary statements supplied with additional information concerning the cut: each component in a generalized statement corresponds with a (nested) procedure call. Since executing a cut amounts to restoring the backtrack stack as it was at the moment of procedure entry (see Definition 3.3(v)) we attach to a statement a stack (or pointer) that constitutes (points to) the substack of the alternatives that should remain open after a cut in the statement is executed. We call this stack the dump stack of the statement, cf. [19].

Definition 3.2. Define the set of generalized statements by

$$GStat = \{\langle s_1, D_1\rangle :\ldots: \langle s_r, D_r\rangle \mid r \in \mathbb{N},\ s_i \in Stat,\ D_i \in Stack,\ i<j \Rightarrow D_i \geqslant_{ss} D_j\}$$

with $\gamma \in GStat$ denoting the empty generalized statement, the set of frames by

$$Frame = \{[g, \sigma] \mid g \in GStat,\ \sigma \in \Sigma\}$$

and the set of stacks by

$$Stack = \{F_1 :\ldots: F_r \mid r \in \mathbb{N},\ F_i = [\langle s_1, D_1\rangle :\ldots: \langle s_q, D_q\rangle, \sigma] \in Frame \text{ such that } F_{i+1} :\ldots: F_r \geqslant_{ss} D_j\}$$

with $E \in Stack$ denoting the empty stack. Here we have $S \geqslant_{ss} S' \Leftrightarrow S'$ is a substack of S.

Fix an action interpretation $I : Action \to \Sigma \to_{part} \Sigma$ that reflects the effect of the execution of an action on a state. (The language $\mathcal{B}$ gains flexibility if actions are allowed to succeed in the one state, while failing in another as is illustrated by the example. Hence we model failure as partiality.) Let TS be the collection of all labeled transition systems with configurations in $Conf = Stack \cup \{\Omega\}$, internal configurations in $Stack \setminus \{E\}$, one final configuration E, undefined configuration Ω, labels in $\Lambda = \Sigma \cup \{\varepsilon\}$, domain of values Σ^{st}, and valuation $\alpha : \Lambda \cup \{E\} \to \Sigma^{st}$ with $\alpha(E) = \varepsilon$, $\alpha(\lambda) = \lambda$.

Next we give the definition of the labeled transition systems underlying the operational semantics. Each declaration d induces a system $\to_d$ in TS determined by its transition relation. The labels are states $\sigma \in \Sigma$ or the empty word $\varepsilon \in \Sigma^*$. For notational convenience we shall write $\to_d$ instead of $\to_d^{\varepsilon}$.

Definition 3.3. Let $d \in Decl$. d induces a labeled transition system in TS with as step relation the smallest subset of $Conf \times \Lambda \times Conf$ such that:

(i) $[\gamma, \sigma] : S \to_d^{\sigma} S$;

(ii) $[\langle \varepsilon, D\rangle : g, \sigma] : S \to_d [g, \sigma] : S$;

(iii) $[\langle a; s, D\rangle : g, \sigma] : S \to_d [\langle s, D\rangle : g, \sigma'] : S$, if $\sigma' = I(a)(\sigma)$ exists,
$[\langle a; s, D\rangle : g, \sigma] : S \to_d S$, otherwise;

(iv) $[\langle \textbf{\textit{fail}}; s, D\rangle : g, \sigma] : S \to_d S$;

(v) $[\langle !; s, D\rangle : g, \sigma] : S \to_d [\langle s, D\rangle : g, \sigma] : D$;

(vi) $[\langle x'; s, D\rangle : g, \sigma] : S \to_d [\langle s', S\rangle : \langle s, D\rangle : g, \sigma] : S$, if $x \leftarrow s \in d$;

(vii) $[\langle (s_1 \textbf{ or } s_2); s, D\rangle : g, \sigma] : S \to_d F_1 : F_2 : S$,
where $F_i = [\langle s_i : s, D\rangle : g, \sigma]$, $i = 1, 2$.

We comment briefly on each of the above transitions (more precisely transition schemes).

(i) If the top frame contains the empty generalized statement, denoted by γ, the computation of this alternative terminates successfully. The local state σ of the frame is delivered as an answer and the frame is popped of the stack. The computation continues with the remaining frames.

(ii) If the left-most component of a generalized statement has become empty (in case a procedure call or the initial statement has terminated), i.e., has format $\langle \varepsilon, D \rangle$, then the statement-dump stack pair is deleted from the frame. The computation continues with the remaining generalized statement.

(iii) In case an action a in the top frame has become active, the action interpretation I is consulted for the effect of a in σ. If $I(a)(\sigma)$ is defined, the state is transformed accordingly. If $I(a)(\sigma)$ is not defined the frame fails and is popped of the stack.

(iv) Execution of ***fail*** amounts to failure of the current alternative. Hence the top frame is popped of the backtracking stack. Control is transferred to the new top frame.

(v) The transition concerning the cut represents removal of alternatives; the top frame continues its execution. Since the dump stack D is a substack of the backtrack stack S, replacing the backtrack stack by the current dump stack indeed amounts, in general, to deletion of frames, i.e., of alternatives. (Note that the right-hand stack is well-formed by definition of *GStat.*)

(vi) A call initiates body replacement. The body is looked up in the declaration d and becomes the active component of the generalized statement in the top frame. This component has its own dump stack, which is (a pointer to) the backtrack stack at call time.

(vii) Execution of an alternative composition yield two new frames: an active frame corresponding to the left component of the ***or***-construct and a suspended frame corresponding to the right component.

Definition 3.4. The operational semantics $\mathcal{O} : \mathcal{B} \to \Sigma \to \Sigma^{st}$ for the backtracking language $\mathcal{B}$ is defined by $\mathcal{O}(d \mid s)(\sigma) = \alpha_d([\langle s, E \rangle, \sigma])$ where $\alpha_d : Conf \to \Sigma^{st}$ is the valuation associated with the labeled transition system induced by d.

Example (*Continued*). Consider the declaration

$$d = x \leftarrow y; !; z : y \leftarrow a; y; a \textbf{ or } (b; y; b \textbf{ or } a) : z \leftarrow c; z \textbf{ or } c.$$

Let us calculate as illustration of the definitions parts of the transition sequence for the statement x;***eoi*** in state *ababad*\$. (Here the state reflects the input buffer. \$ represents acceptance; all other states represent rejection.) The interpretation of the actions a, b, c and ***eoi*** is as described previously: for $\alpha \in \{a, b, c\}$ we assume $I(\alpha)(\alpha w) = w$, $I(\alpha)$ is undefined otherwise and $I(\textbf{eoi})(\$) = \$$, $I(\textbf{eoi})$ fails, i.e. is

undefined, otherwise. Note $ababad \notin \mathscr{L}$ for $d \notin \{a, b, c\}$.

$[\langle x;\textbf{\textit{eoi}}, E\rangle, ababad\$]$

(1) $\rightarrow$

$[\langle y;!;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

(2) $\rightarrow$

$[\langle a;y;a \textbf{ or } (b;y;b \textbf{ or } a), E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

(3) $\rightarrow$

$[\langle a;y;a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

$[\langle b;y;b \textbf{ or } a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

(4) $\rightarrow$

$[\langle y;a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, babad\$]$

$[\langle b;y;b \textbf{ or } a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

$\rightarrow$

$\vdots$

$\rightarrow$

$[\langle \varepsilon, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, d\$]$

$[\langle a, \#1\rangle{:}\langle a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, babad\$]$

$[\langle b;y;b \textbf{ or } a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

(5) $\rightarrow$

$[\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, d\$]$

$[\langle a, \#1\rangle{:}\langle a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, babad\$]$

$[\langle b;y;b \textbf{ or } a, E\rangle{:}\langle !;z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, ababad\$]$

(6) $\rightarrow$

$[\langle z, E\rangle{:}\langle \textbf{\textit{eoi}}, E\rangle, d\$]$

$\rightarrow$

$\vdots$

$\rightarrow$

E

Here $\#1$ denotes a pointer into the appropriate substack. Transitions (1) and (2) follow the scheme of Definition 3.3(vi), and create new components in the generalized statements. Transition (3) shows how the alternatives are distributed according to Definition 3.3(vii). Since the action a succeeds in state $ababad\$$, yielding $babad\$$, the first clause of Definition 3.3(iii) is applicable at transition (4). At transition (5) the procedure call for y has terminated. The corresponding component is deleted. At transition (6) one can see the effect of evaluation of the cut: execution of the cut amounts to removal of the two lowest frames.

4. Denotational semantics for $\mathscr{B}$

In this section we present a direct denotational semantics for the backtracking language $\mathscr{B}$. This semantics is similar to the semantics for PROLOG with cut in [19]. The standard technique using environments is used to establish the meaning of procedures (cf. [6, 27].) First, meaning is given to statements in arbitrary environ-

ments. Next, for each declaration an environment transformation is defined that captures the notion of body replacement. Finally, the denotational semantics of a $\mathcal{B}$-program evaluates the program body in the environment of the least fixed point of the transformation corresponding to the declaration of the program.

Let us first of all give some explanation of the functionality of the statement evaluator $[\![\cdot]\!]_s$ which establishes the meaning of statements, postponing for a moment the motivation of the interpretation of the statements itself. It is clear that the cut forces to supply sequential information to the semantical mapping $[\![\cdot]\!]_s$, for the cut selects the *first* alternative, i.e., state, and throws away others. For example, we need to distinguish $[\![(a \textbf{ or } b);!]\!]_s$ from $[\![a]\!]_s$, since operationally $(a \textbf{ or } b);!$ and a behave differentially in the context of yet another alternative c: $\mathcal{O}((a \textbf{ or } b);! \textbf{ or } c)(\sigma) = \sigma_a$ while $\mathcal{O}(a \textbf{ or } c) = \sigma_a\sigma_c$. One way or the other is should be made explicit that a cut has taken place, and that alternatives should not be appended consequently. We introduce the special marker δ for the purpose of signaling the execution of a cut !. For this we extend our stream domain with streams possibly ending in δ, i.e., to the domain Σ_δ^{st}. Recalling the use of environments to handle procedure calls we end up with the functionality $[\![\cdot]\!]_s : Stat \to Env \to \Sigma \to \Sigma_\delta^{st}$ where the collection of environments $Env = Proc \to \Sigma \to \Sigma_\delta^{st}$. We shall find it convenient to have also available an evaluator $[\![\cdot]\!]_e : EStat \to Env \to \Sigma \to \Sigma_\delta^{st}$ to handle elementary statements separately.

Next we define some auxiliary operators that will be used in the semantical mapping. Again the action interpretation I will be used to establish the meaning of actions. However, here we consider I as a mapping in $Action \to \Sigma \to \Sigma_\delta^{st}$. Here we have $I(a)(\sigma) = \varepsilon$ when $I(a)(\sigma)$ was not defined before. We shall use the mappings $fail, cut : \Sigma \to \Sigma_\delta^{st}$ to denote ***fail*** and !, respectively. We put $fail(\sigma) = \varepsilon$ and $cut(\sigma) = \sigma\delta$. For the failure statement ***fail*** delivers no answer in any state. The cut operator ! does not transform the state but has a side-effect expressed by the cut marker δ. Also we have $id : \Sigma \to \Sigma_\delta^{st}$ with $id(\sigma) = \sigma$. Let $M = \Sigma \to \Sigma_\delta^{st}$. To give meaning to the ***or***-construct we define a concatenation operator $\cdot : M \times M \to M$ by $(\psi \cdot \phi)(\sigma) = \psi(\sigma) \cdot \phi(\sigma)$. For the sequential composition we define a composition operator $\circ : M \times M \to M$ by $(\psi \circ \phi)(\sigma) = \hat{\phi}(\psi(\sigma))$ where $\hat{\phi} : \Sigma_\delta^{st} \to \Sigma_\delta^{st}$ is the homomorphic extension of ϕ, i.e., $\hat{\phi}(\xi) = \xi$ for $\xi \in \{\varepsilon, \perp, \delta\}$ and $\hat{\phi}(\sigma\tau) = \phi(\sigma) \cdot \hat{\phi}(\tau)$. Finally the hiding or uncut operator $\cdot \backslash \delta : \Sigma_\delta^{st} \to \Sigma_\delta^{st}$ is defined by $\tau \backslash \delta = \theta$ if $\tau = \theta\delta$ and $\tau \backslash \delta = \tau$ otherwise.

Definition 4.1.

(i) $[\![\cdot]\!]_e : EStat \to Env \to \Sigma \to \Sigma_\delta^{st}$,
$[\![a]\!]_e\eta = I(a)$,
$[\![\textbf{fail}]\!]_e\eta = fail$,
$[\![!]\!]_e\eta = cut$,
$[\![s_1 \textbf{ or } s_2]\!]_e\eta = [\![s_1]\!]_s\eta \cdot [\![s_2]\!]_s\eta$,
$[\![x]\!]_e\eta = \eta x$;

(ii) $[\![\cdot]\!]_s : Stat \to Env \to \Sigma \to \Sigma_\delta^{st}$,
$[\![\varepsilon]\!]_s\eta = id$,
$[\![e;s]\!]_s\eta = [\![s]\!]_s\eta \circ [\![e]\!]_e\eta$;

(iii) $\Phi : Decl \to Env \to Env$,
$\Phi d\eta = \lambda x.\lambda\sigma.[\![s]\!]_s\eta\sigma \setminus \delta$ if $x \leftarrow s \in d$;

(iv) $\mathcal{D} : \mathcal{B} \to \Sigma \to \Sigma_\delta^{st}$,
$\mathcal{D}(d \mid s) = [\![s]\!]_s\eta_d$, where η_d is the least fixed point of $\Phi(d)$.

The stream $[\![e]\!]_e\eta\sigma$ denotes the meaning of the elementary statement e in the environment η and state σ. So if e is an action $a \in Action$ the action interpretation I is consulted. In case of ***fail*** or ! the semantical counterparts *fail* and *cut* serve as denotations. The ***or***-construct is denoted by the concatenation operator. We use environments to establish the meaning of procedure names.

The semantical mapping $[\![\cdot]\!]_s$ on *Stat* is now straightforward: The empty statement has no effect on the input. So $[\![\varepsilon]\!]_s\eta\sigma = \sigma$ as is expressed by the mapping *id*. Sequential composition is replaced by the composition operator: For $\sigma \in \Sigma$ first $[\![e]\!]_e\eta\sigma$ is calculated, say yielding a stream τ. This τ is pipelined to $[\![s]\!]_s\eta$. The sequence of alternatives, i.e., the states in τ are processed one by one by s and the results concatenated together (taking into account the markers $\perp$ and δ).

The environment transformation Φ captures the notion of body replacement for procedure calls. For a procedure name x with body s in a declaration d, denoted by $x \leftarrow s \in d$, the outcome of $\Phi d\eta x$ in state σ is the meaning of the statement s in environment η and state σ, but with a possible δ hided. Since the special symbol δ signals that a cut at the particular call level has taken place, a δ at the level of s should not be propagated to the level of x (cf. Section 3). As a consequence we do not have in general that the meaning $[\![x]\!]_s$ of the call x and $[\![s]\!]_s$ of its body s coincide.

Finally the semantics of a $\mathcal{B}$-program $d \mid s$ is established by evaluating the statement s in an environment induced by the declaration d, viz. the least fixed point η_d of $\Phi(d)$. This semantics is well-defined by virtue of the next lemma.

Lemma 4.2. *$[\![\cdot]\!]_e$, $[\![\cdot]\!]_s$ and Φ are continuous in η.*

Proof. Left to the reader. □

The denotational semantics for $\mathcal{B}$ uses the technique of environment transformation as in [19]. Alternatively we could use a simultaneous fixpoint construction for finitely many procedure names if we restrict to so-called closed programs [6].

Remark. The least fixed point defined in Definition 4.1(iv) can be obtained as the least upperbound of a chain of iterations $\langle \eta_{d,i} \rangle_i$, with $\eta_{d,i}$ defined by $\eta_{d,0} = \lambda x.\lambda\sigma.\perp$ and $\eta_{d,i+1} = \Phi d\eta_{d,i}$. From the continuity of $[\![\cdot]\!]_s$ we derive $[\![s]\!]_s\eta_d = lub_i([\![s]\!]_s\eta_{d,i})$.

Examples. Consider the actions *one*, *two* $\in$ *Action* with

$$I(one)(\sigma) = \sigma_1, \qquad I(two)(\sigma) = \sigma_2$$

for all $\sigma \in \Sigma$ (and some $\sigma_1, \sigma_2 \in \Sigma$). Choose $\sigma \in \Sigma$ and $d \in Decl$. Then we have

$$[\![d|\textit{one or two}]\!]_{\mathscr{B}}\sigma = [\![\textit{one or two}]\!]_{\mathscr{S}}\eta_d\sigma$$
$$= [\![\textit{one}]\!]_{\mathscr{E}}\eta_d\sigma \cdot [\![\textit{two}]\!]_{\mathscr{E}}\eta_d\sigma = \sigma_1\sigma_2$$

and

$$[\![d|(\textit{one or two});!]\!]_{\mathscr{B}}\sigma = ([\![!]\!]_{\mathscr{S}}\eta_d)^{\wedge}([\![\textit{one or two}]\!]_{\mathscr{S}}\eta_d\sigma)$$
$$= ([\![!]\!]_{\mathscr{S}}\eta_d)^{\wedge}(\sigma_1\sigma_2) = [\![!]\!]_{\mathscr{S}}\eta_d\sigma_1 \cdot [\![!]\!]_{\mathscr{S}}\eta_d\sigma_2$$
$$= \sigma_1\delta \cdot \sigma_2\delta = \sigma_1\delta$$

(where $(\cdot)^{\wedge}$ denotes the homomorphic extension operator).

Similarly, the statements ***fail*** and !;***fail*** have different meanings. For on the one hand we have $[\![d|\textit{fail}]\!]_{\mathscr{B}} = \lambda\sigma.\varepsilon$ while on the other hand $[\![d|!;\textit{fail}]\!]_{\mathscr{B}} = \lambda\sigma.\delta$. However, the operational semantics of ***fail*** and !;***fail*** do coincide:

$$\mathscr{O}(d|\textit{fail}) = \mathscr{O}(d|!;\textit{fail}) = \lambda\sigma.\varepsilon,$$

since

$$[\langle !;\textit{fail}, E\rangle, \sigma] \rightarrow_d [\langle \textit{fail}, E\rangle, \sigma] \rightarrow_d E.$$

In the next section we shall introduce an intermediate semantics $\mathscr{I}$ that is an extension of the operational semantics $\mathscr{O}$ but that will distinguish between ***fail*** and !;***fail***.

5. Equivalence of $\mathscr{O}$ and $\mathscr{D}$

As we have seen in the previous section the compositional character of the denotational semantics implies the signaling of cuts that have been encountered on the current level. The operational semantics does not need a token indicating the execution of a cut, since alternatives are maintained dynamically on the backtrack stack. Operationally, alternatives not occurring in the dump stack are in effect removed from the backtrack stack; denotationally, the marker δ will absorb their denotations. In order to prove the correctness of the denotational semantics with respect to the operational one we introduce a transition-system-based intermediate semantics $\mathscr{I}$ which is an extension of the operational semantics and which is able to signal cuts occurring at the top level.

The definition of the intermediate semantics follows that of the operational one. For each declaration we define a transition system (using the same clauses as before) and the intermediate semantics of a program $d|s$ in state σ is the value of the maximal transition sequence with respect to d from the initial state corresponding to s and σ. The new thing here is the stack (or stack marker) Δ. Δ is a final configuration contributing δ to the value of a transition sequence. The initial configuration associated with a statement-state pair s, σ in the setting of the intermediate semantics is the frame $[\langle s, \Delta\rangle, \sigma]$. For if a cut is encountered in s the current dump stack, i.e. Δ, replaces the backtrack stack, yielding a stack of the format $F{:}\Delta$. All the configurations in the transition sequence from there on will be

of the format $S{:}\Delta$, i.e., are marked with Δ. If the maximal transition sequence ended before (with respect to $\mathcal{O}$) in E, it now (with respect to $\mathcal{I}$) ends in Δ yielding a stream ending with δ. If the computation was infinite, it still is. Of course, it is possible for a transition sequence to have E as a final configuration with respect to $\mathcal{I}$. This is the case if no cut is encountered at the top level.

The transition schemes for the intermediate case are exactly the same as before in Section 3. (The changes are implicit in the range of the metavariables.) In particular, the current backtrack stack is taken as dump stack in case of body replacement for a procedure call. This makes the definition of stacks slightly complicated, since additional to the pointer property, i.e., the dump stacks being substacks of the backtrack stack, we have to take care that once a stack is marked with Δ this marker will not be removed by any dump stack. Thus we forbid generalized statement of the format $\langle s_1, \Delta\rangle{:}\langle s_2, E\rangle{:}\langle s_3, \Delta\rangle$ while we still allow the format $\langle s_2, E\rangle{:}\langle s_3, \Delta\rangle$. For, in the former case the Δ introduced by a cut in s_1 can be overwritten by a cut in s_2. In the latter situation this is not possible.

Definition 5.1. Define the set of generalized statements by

$$GStat' = \{\langle s_1, D_1\rangle{:}\ldots{:}\langle s_r, D_r\rangle \mid r \in \mathbb{N},\ s_i \in Stat,\ D_i \in Stack',\ i<j \Rightarrow D_i \geqslant'_{ss} D_j\},$$

the set of frames by

$$Frame' = \{[g, \sigma] \mid g \subset GStat',\ \sigma \subset \Sigma\}$$

and the set of stacks by

$$\begin{aligned} Stack' = \{F_1{:}\ldots{:}F_r \mid r \in \mathbb{N},\ F_i = [\langle s_1, D_1\rangle{:}\ldots{:}\langle s_q, D_q\rangle, \sigma] \in Frame' \\ \text{such that } F_{i+1}{:}\ldots{:}F_r \geqslant_{ss} D_j\} \\ \cup \{F_1{:}\ldots{:}F_r{:}\Delta \mid r \in \mathbb{N},\ F_i = [\langle s_1, D_1\rangle{:}\ldots{:}\langle s_q, D_q\rangle, \sigma] \in Frame' \\ \text{such that } F_{i+1}{:}\ldots{:}F_r{:}\Delta \geqslant_{ss} D_j\} \end{aligned}$$

(with $S \geqslant'_{ss} S' \Leftrightarrow S, S' \in (Frame')^*$ & S' is a substack of S, or $S, S' \in (Frame')^*.\Delta$ and S' is a substack of S, or $S' = \Delta$).

Let TS' be the collection of all labeled transition systems with configurations in $Conf' = Stack' \cup \{\Omega\}$, internal configurations in $Stack' \setminus \{E, \Delta\}$, final configurations in $\{E, \Delta\}$, undefined configuration Ω, labels in $\Lambda = \Sigma \cup \{\varepsilon\}$, domain of values Σ^{st}_{δ}, and valuation $\alpha' : \Lambda \cup \{E, \Delta\} \to \Sigma^{st}_{\delta}$ with $\alpha'(\lambda) = \lambda$, $\alpha'(E) = \varepsilon$, $\alpha'(\Delta) = \delta$.

As mentioned earlier the transition system $\rightarrow_d$ induced by a declaration d in TS' uses exactly the same transition schemes as the transition system $\rightarrow_d$ in TS in Definition 3.3.

Definition 5.2. Let $d \in Decl.$ d induces a deterministic transition system in TS' with as step relation the smallest subset of $Conf' \times \Lambda \times Conf'$ such that

(i) $[\gamma, \sigma]:S \to_\delta^\sigma S;$

(ii) $[\langle \varepsilon, D\rangle:g, \sigma]:S \to_d [g, \sigma]:S;$

(iii) $[\langle a;s, D\rangle:g, \sigma]:S \to_d [\langle s, D\rangle:g, \sigma']:S$, if $\sigma' = I(a)\sigma$ exists,
$[\langle a;s, D\rangle:g, \sigma]:S \to_d S$, otherwise;

(iv) $[\langle \textbf{\textit{fail}};s, D\rangle:g, \sigma]:S \to_d S;$

(v) $[\langle !;s, D\rangle:g, \sigma]:S \to_d [\langle s, D\rangle:g, \sigma]:D;$

(vi) $[\langle x';s, D\rangle:g, \sigma]:S \to_d [\langle s', S\rangle:\langle s, D\rangle:g, \sigma]:S$, if $x \leftarrow s \in d;$

(vii) $[\langle (s_1 \textbf{ or } s_2);s, D\rangle:g, \sigma]:S \to_d F_1:F_2:S,$
where $F_i = [\langle s_i;s, D\rangle:g, \sigma]$, $i = 1, 2.$

The intermediate semantics is constructed similar to the operational semantics in Section 3. The small differences are due to the incorporation of δ and Δ.

Definition 5.3. The intermediate semantics $\mathscr{I}: \mathscr{B} \to \Sigma \to \Sigma_\delta^{\mathrm{st}}$ for the backtracking language $\mathscr{B}$ is defined by $\mathscr{I}(d\,|\,s)\sigma = \alpha'_d([\langle s, \Delta\rangle, \sigma])$ where $\alpha'_d: Conf' \to \Sigma_\delta^{\mathrm{st}}$ is the valuation associated with the labeled transition system in TS' induced by d.

The remainder of the section is devoted to the relationship of the intermediate semantics on the one hand and the operational and denotational semantics on the other. First, we relate $\mathscr{O}$ and $\mathscr{I}$. The operational and intermediate semantics are the same modulo δ, i.e., $\mathscr{O} = \mathscr{I}\backslash\delta$ (where $\cdot\backslash\delta$ is the hiding operator of Section 4), since the underlying transition systems are the same modulo Δ. Secondly, we consider the collection of configurations $Conf'$ as the union of an increasing sequence of subsets of configurations $Conf'_0 \subseteq Conf'_1 \subseteq Conf'_2 \cdots$ where in the subset $Conf'_k$ we allow at most $k+1$ call levels. Following the remark in Section 2, these subsets induce a chain of restricted transition systems and consequently a chain of restricted intermediate semantics. It will be shown to hold $\mathscr{I} = lub_k(\mathscr{I}_k)$. The denotational semantics $\mathscr{D}$ can also be represented on as a chain by continuity of the functions involved, (cf. lemma 4.2). Say $\mathscr{D} = lub_k(\mathscr{D}_k)$. Most of the effort is devoted to obtain for each k the equality $\mathscr{I}_k = \mathscr{D}_k$, since we have to derive the compositionality of the transition-system-oriented semantics $\mathscr{I}_k$. Once this is established we immediately derive $\mathscr{I} = \mathscr{D}$ by continuity arguments. As a corollary we have the relationship $\mathscr{O} = \mathscr{D}\backslash\delta$ between the operational and denotational semantics.

Before we relate $\mathscr{O}$ and $\mathscr{I}$ we first give an auxiliary operator $\cdot\backslash\Delta$ that renames the marker Δ in the empty stack E (analogously to $\cdot\backslash\delta$ that renames δ into ε). We define $\cdot\backslash\Delta: Conf' \to Conf'$ by $E\backslash\Delta = E$, $\Omega\backslash\Delta = \Omega$ and $\Delta\backslash\Delta = E$, and if $F = [\langle s_1, D_1\rangle:\ldots:\langle s_r, D_r\rangle, \sigma]$ then $(F:S)\backslash\Delta = \bar{F}:(S\backslash\Delta)$ where

$$\bar{F} = [\langle s_1, \bar{D}_1\rangle:\ldots:\langle s_r, \bar{D}_r\rangle, \sigma] \quad \text{and} \quad \bar{D}_i = D_i\backslash\Delta.$$

$\cdot\backslash\Delta$ is well-defined. It follows from inspection of the transition schemes that $C \to_d C'$ with respect to $\mathscr{I}$ implies $C\backslash\Delta \to_d C'\backslash\Delta$ with respect to $\mathscr{O}$.

Theorem 5.4. *For each program $d\,|\,s$ in $\mathcal{B}$, it holds that $\mathcal{O}(d\,|\,s) = \lambda\sigma.\mathcal{I}(d\,|\,s)\sigma\backslash\delta$.*

Proof. Let $d\,|\,s \in \mathcal{B}$ and $\sigma \in \Sigma$. First we prove by fixpoint induction: $\alpha'_d(C)\backslash\delta = \alpha_d(C\backslash\Delta)$, where α'_d and α_d are the valuations of the transition systems induced by d with respect to $\mathcal{I}$ and $\mathcal{O}$, respectively. For $C \in \{\Omega, E, \Delta\}$ this is clear. Otherwise $C \to_d^{\lambda} C'$ with respect to $\mathcal{I}$ for some label λ and configuration $C \in Conf'$. Then

$$\begin{aligned}\alpha'_d(C)\backslash\delta &= (\alpha'(\lambda)\cdot\alpha'_d(C'))\backslash\delta\\ &= \alpha(\lambda\cdot(\alpha'_d(C')\backslash\delta) \quad \text{since } \lambda \neq \delta\\ &= \alpha(\lambda)\cdot\alpha_d(C'\backslash\Delta) \quad \text{by induction hypothesis}\\ &= \alpha_d(C\backslash\Delta) \quad \text{since } C\backslash\Delta \to_d^{\lambda} C'\backslash\Delta \text{ with respect to } \mathcal{O}.\end{aligned}$$

So

$$\begin{aligned}\mathcal{I}(d\,|\,s)\sigma\backslash\delta &= \alpha'_d([\langle s, \Delta\rangle, \sigma])\backslash\delta = \alpha_d([\langle s, \Delta\rangle, \sigma]\backslash\Delta) = \alpha_d([\langle s, E\rangle, \sigma])\\ &= \mathcal{O}(d\,|\,s)\sigma. \qquad \square\end{aligned}$$

Next we represent the intermediate semantics as a chain of restricted intermediate semantics. We introduce subsets $Conf'_k$ of $Conf'$ and restrictions $\to_{d,k}$ of $\to_d$ to $Conf'_k$ in TS'. The subscript k indicates that generalized statements occurring in stacks in $Conf'_k$ consist at most of $k+1$ components $\langle s, D\rangle$. Intuitively, we impose a maximum, viz. k, on the number of inner calls.

Definition 5.5. The weight function $w: GStat', Frame', Stack', Conf' \to \mathbb{N}$ is defined by

$$w(\langle s_1, D_1\rangle{:}\ldots{:}\langle s_r, D_r\rangle) = r \dot{-} 1,$$

$$w([g, \sigma]) = w(g), \qquad w(E) = 0, \qquad w(\Omega) = 0,$$

$$w(F{:}S) = \max\{w(F), w(S)\},$$

where $\dot{-}$ denotes the monus, i.e., subtraction in $\mathbb{N}$.

We put

$$Stack'_k = \{S \in Stack' \,|\, w(S) \leqslant k\}$$

and

$$Conf'_k = \{C \in Conf' \,|\, w(C) \leqslant k\}.$$

So we have $Conf'_k = Stack'_k \cup \{\Omega\}$. For $d \in Decl$ and $i \in \mathbb{N}$ we write $\to_{d,k}$ for the restriction of $\to_d$ to $Conf'_k$ in TS'. So in particular we have for $x' \leftarrow s' \in d$:

(i) $[\langle x'; s, D\rangle{:}g, \sigma]{:}S \to_{d,k} [\langle s', S\rangle{:}\langle s, D\rangle{:}g, \sigma]{:}S$, if $w(\langle x'; s, D\rangle{:}g) < k$,

(ii) $[\langle x'; s, D\rangle{:}g, \sigma]{:}S \to_{d,k} \Omega$, if $w(\langle x'; s, D\rangle{:}g) = k$,

(iii) $[\langle x'; s, D\rangle{:}g, \sigma]{:}S$ admits no transition, if $w(\langle x'; s, D\rangle{:}g) > k$.

Definition 5.6. For $k \in \mathbb{N}$ we define $\mathscr{I}_k : \mathscr{B} \to \Sigma \to \Sigma_\delta^{st}$ by $\mathscr{I}_k(d\,|\,s)\sigma = \alpha'_{d,k}([\langle s, \Delta\rangle, \sigma])$ where $\alpha'_{d,k}$ is the valuation of the labeled transition system $\to_{d,k}$ in TS', which is the restriction of $\to_d$ to $Stack_k \setminus \{E, \Delta\}$.

According to the general remarks in Section 2, based on the continuity of the mapping $\lambda T.\alpha_T \in TS' \to Conf' \to \Sigma_\delta^{st}$ that assigns to a transition system its associated valuation, we have that the chain $\langle \to_{d,k}\rangle_k$ approximates $\to_d$. Hence the intermediate semantics $\mathscr{I}$ is the least upperbound of the chain of restricted semantics $\langle \mathscr{I}_k\rangle_k$.

Theorem 5.7. *For all programs $d\,|\,s$ in $\mathscr{B}$ and all states $\sigma \in \Sigma$,*

$$\mathscr{I}(d\,|\,s)\sigma = lub_k(\mathscr{I}_k(d\,|\,s)\sigma).$$

Proof. Let $d\,|\,s \in \mathscr{B}$ and all states $\sigma \in \Sigma$: Since $\to_d$ is the least upper bound of the chain $\langle \to_{d,k}\rangle_k$ in TS' we have

$$\alpha'_d([\langle s, \Delta\rangle, \sigma]) = lub_k(\alpha'_{d,k}([\langle s, \Delta\rangle, \sigma])),$$

i.e. $\mathscr{I}(d\,|\,s)\sigma = lub_k(\mathscr{I}_k(d\,|\,s)\sigma)$. □

The restricted intermediate semantics $\mathscr{I}_k$ will be related to "restricted" denotational semantics $\mathscr{D}_k$ to be defined in a moment. In these semantics we intuitively have also a bound on the nesting of procedure calls as we have with respect to the restricted intermediate semantics. We shall have $\mathscr{D}_k(d\,|\,s)\sigma = [\![s]\!]_S \eta_{d,k}\sigma$ where $\eta_{d,k}$ is the kth iteration of Φd. The environment $\eta_{d,k}$ expresses k times a simultaneous application of the dynamic copy rule for the procedure names involved. In particular $\eta_{d,0}$ is the least environment $\lambda x.\lambda\sigma.\bot$ assigning to x the uninforming $\bot$ since no body replacement has taken place.

Definition 5.8. For $d \in Decl$, we define the environments $\eta_{d,k}$ inductively by $\eta_{d,0} = \lambda x.\lambda\sigma.\bot$ and $\eta_{d,k+1} = \Phi d\eta_{d,k}$. The restricted denotational semantics $\mathscr{D}_k : \mathscr{B} \to \Sigma \to \Sigma_\delta^{st}$ is defined by $\mathscr{D}_k(d\,|\,s) = [\![s]\!]_S \eta_{d,k}$.

Next we want to relate the restricted intermediate and denotational semantics. Theorem 5.13, stating $\mathscr{I}_k(d\,|\,s) = \mathscr{D}_k(d\,|\,s)$ for programs $d\,|\,s$ in $\mathscr{B}$, will be established by induction on k and the statement s. In order to prove the equality for statements of the format s_1 ***or*** s_2 and $e;s$ and for procedures x (in case $k > 0$) we need some further elaboration in $\mathscr{I}_k$. For this we shall use the fact that the transition systems $\to_{d,k}$ are terminating or Noetherian, i.e., there exists no infinite sequence $(C_i)_i$ in $Conf'$ such that $C_i \to_{d,k} C_{i+1}$. This property allows us to reason by induction on the length of the computations involved, and moreover it enables us to switch smoothly between the restricted intermediate semantics $\mathscr{I}_k$ and its underlying transition system $\to_{d,k}$. In contrast, the termination property is not shared by the limit $\to_d$ underlying $\mathscr{I}$.

Lemma 5.9. *All the transition systems* $\rightarrow_{d,k}$ *are terminating.*

Proof. See [13]. □

From the lemma it follows that for $C \in Conf'_k$ we have either $C \twoheadrightarrow^{\theta}_{d,k} E$ or $C \twoheadrightarrow^{\theta}_{d,k} \Delta$ or $C \twoheadrightarrow^{\theta}_{d,k} \Omega$ for some suitable $\theta \in \Sigma^*$. Moreover,

$$C \twoheadrightarrow^{\theta}_{d,k} E \Leftrightarrow \alpha'_{d,k}(C) = \theta,$$
$$C \twoheadrightarrow^{\theta}_{d,k} \Delta \Leftrightarrow \alpha'_{d,k}(C) = \theta\delta,$$
$$C \twoheadrightarrow^{\theta}_{d,k} \Omega \Leftrightarrow \alpha'_{d,k}(C) = \theta\perp.$$

So

$$\mathcal{I}_k(d\,|\,s)\sigma = \theta\zeta \Leftrightarrow [\langle s, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k} Z$$

with $\zeta \in \{\varepsilon, \perp, \delta\}$ and $Z \in \{E, \Omega, \Delta\}$ appropriate. (Here the double arrow $\twoheadrightarrow$ denotes the reflexive and transitive closure of the arrow $\rightarrow$ with the labels handled in the obvious way.)

Next we turn to the compositionality of $\mathcal{I}_k$ with respect to the ***or***-construct. We define an auxiliary operator $\cdot \lhd S$ to insert alternative frames below stacks. Let $\bar{S} \in Stack'\backslash\{\Delta\}$. We define $\cdot \lhd \bar{S}: Conf' \rightarrow Conf'$ by $E \lhd \bar{S} = \bar{S}$, $\Omega \lhd \bar{S} = \Omega$, $\Delta \lhd \bar{S} = \Delta$, and if $F = [\langle s_1, D_1\rangle : \ldots : \langle s_r, D_r\rangle, \sigma]$ then $(F:S) \lhd \bar{S} = \bar{F}:(S \lhd \bar{S})$ where

$$\bar{F} = [\langle s_1, \bar{D}_1\rangle : \ldots : \langle s_r, \bar{D}_r\rangle, \sigma] \quad \text{and} \quad \bar{D}_i = D_i \lhd \bar{S}.$$

It is readily checked that $S \lhd \bar{S} \in Stack'$, $\cdot \lhd \bar{S}$ is well-defined and

$$(C \twoheadrightarrow^{\theta}_{d,k} C') \;\&\; (w(\bar{S}) \leqslant k) \Rightarrow C \lhd \bar{S} \twoheadrightarrow^{\theta}_{d,k} C' \lhd \bar{S}.$$

(Notice the similarity between the definition of $\cdot$ on $\Sigma^{\mathrm{st}}_{\delta}$ and $\lhd$ on *Conf*. In particular, we have $\delta \cdot y = \delta$ and $\Delta \lhd \bar{S} = \Delta$.)

Lemma 5.10. *Let* $k \in \mathbb{N}$, $d \in Decl$ *and* $s_1, s_2 \in Stat$. *Then we have*

$$\mathcal{I}_k(d\,|\,s_1 \;\textbf{or}\; s_2) = \mathcal{I}_k(d\,|\,s_1) \cdot \mathcal{I}_k(d\,|\,s_2).$$

Proof. Let $\sigma \in \Sigma$. Suppose $[\langle s_1, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Delta$ (with respect to $\rightarrow_{d,k}$). Then

$$[\langle s_1, \Delta\rangle, \sigma] : [\langle s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Delta,$$

since

$$[\langle s_1, \Delta\rangle, \sigma] \lhd [\langle s_2, \Delta\rangle, \sigma] = [\langle s_1, \Delta\rangle, \sigma] : [\langle s_2, \Delta\rangle, \sigma]$$

and

$$\Delta \lhd [\langle s_2, \Delta\rangle, \sigma] = \Delta.$$

So

$$[\langle s_1 \;\textbf{or}\; s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Delta.$$

Analogously, if $[\langle s_1, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Omega$, then $[\langle s_1 \;\textbf{or}\; s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Omega$, and if $[\langle s_1, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} E$ and $[\langle s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_2}_{d,k} Z$, then $[\langle s_1 \;\textbf{or}\; s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1\theta_2}_{d,k} Z$.

Suppose $\mathscr{I}_k(d\,|\,s_1)\sigma = \theta_1\zeta_1$ and $\mathscr{I}_k(d\,|\,s_2)\sigma = \theta_2\zeta_2$ with $\theta_1, \theta_2 \in \Sigma^*$ and $\zeta_1, \zeta_2 \in \{\varepsilon, \bot, \delta\}$. Then we have $[\langle s_1, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} Z_1$ and $[\langle s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_2}_{d,k} Z_2$ with $Z_1, Z_2 \in \{E, \Omega, \Delta\}$ corresponding to ζ_1, ζ_2. By the above: if $\zeta_1 = \varepsilon$, then

$$Z_1 = E,$$

$$[\langle s_1 \textbf{ or } s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1\theta_2}_{d,k} Z_2,$$

$$\mathscr{I}_k(d\,|\,s_1 \textbf{ or } s_2)\sigma = \theta_1\theta_2\zeta_2 = \theta_1 \cdot \theta_2\zeta_2;$$

if $\zeta_1 = \bot$, then

$$Z_1 = \Omega,$$

$$[\langle s_1 \textbf{ or } s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Omega,$$

$$\mathscr{I}_k(d\,|\,s_1 \textbf{ or } s_2)\sigma = \theta_1\bot = \theta_1\bot \cdot \theta_2\zeta_2;$$

if $\zeta_1 = \delta$, then

$$Z_1 = \Delta,$$

$$[\langle s_1 \textbf{ or } s_2, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_1}_{d,k} \Delta,$$

$$\mathscr{I}_k(d\,|\,s_1 \textbf{ or } s_2)\sigma = \theta_1\delta = \theta_1\delta \cdot \theta_2\zeta_2.$$

So in all cases we have

$$\mathscr{I}_k(d\,|\,s_1 \textbf{ or } s_2)\sigma = \mathscr{I}_k(d\,|\,s_1)\sigma \cdot \mathscr{I}_k(d\,|\,s_2)\sigma. \qquad \square$$

In Lemma 5.11, where we focus on body replacement with respect to the restricted intermediate semantics we need yet two other auxiliary operators. The hiding operator $\cdot\backslash\Delta$ that renames Δ into the empty stack E has been used previously. The operator $\cdot:\langle\varepsilon, \Delta\rangle$ introduces in all the frames of a stack a dummy frame, which can be regarded as a rudiment of $\langle x, \Delta\rangle$ after the x has been expanded.

Recall $\cdot\backslash\Delta : Conf' \to Conf'$ defined by $E\backslash\Delta = E$, $\Omega\backslash\Delta = \Omega$ and $\Delta\backslash\Delta = E$, and if $F = [\langle s_1, D_1\rangle{:}..{:}\langle s_r, D_r\rangle, \sigma]$ then $(F{:}S)\backslash\Delta = \bar{F}{:}(S\backslash\Delta)$ where

$$\bar{F} = [\langle s_1, \bar{D}_1\rangle{:}..{:}\langle s_r, \bar{D}_r\rangle, \sigma] \quad \text{and} \quad \bar{D}_i = D_i\backslash\Delta.$$

$\cdot\backslash\Delta$ is well-defined, and for $S \in Stack'_k$ we have $S\backslash\Delta \in Stack'_k$. It follows from inspection of the clauses of $\to_{d,k}$ that $C \twoheadrightarrow^{\theta}_{d,k} C'$ implies $C\backslash\Delta \twoheadrightarrow^{\theta}_{d,k} C'\backslash\Delta$. We define $\cdot:\langle\varepsilon, \Delta\rangle: Conf' \to Conf'$ by $E:\langle\varepsilon, \Delta\rangle = E$, $\Omega:\langle\varepsilon, \Delta\rangle = \Omega$, $\Delta:\langle\varepsilon, \Delta\rangle = \Delta$ and if $F = [\langle s_1, D_1\rangle{:}..{:}\langle s_r, D_r\rangle, \sigma]$ then $(F{:}S){:}\langle\varepsilon, \Delta\rangle = \bar{F}{:}(S{:}\langle\varepsilon, \Delta\rangle)$ where

$$\bar{F} = [\langle s_1, \bar{D}_1\rangle{:}..{:}\langle s_r, \bar{D}_r\rangle{:}\langle\varepsilon, \Delta\rangle, \sigma] \quad \text{and} \quad \bar{D}_i = D_i{:}\langle\varepsilon, \Delta\rangle.$$

Clearly $\cdot:\langle\varepsilon, \Delta\rangle$ is well defined. Again it follows from inspection of the clauses of the intermediate transition system $\to_{d,k}$ that $C \twoheadrightarrow^{\theta}_{d,k} C'$ implies $C{:}\langle\varepsilon, \Delta\rangle \twoheadrightarrow^{\theta}_{d,k} C'{:}\langle\varepsilon, \Delta\rangle$.

Lemma 5.11. *Let $k \in \mathbb{N}$, $d \in Decl$, $x \in Proc$ and $s \in Stat$ such that $x \leftarrow s \in d$. Then it holds that*

$$\mathscr{I}_{k+1}(d\,|\,x) = \lambda\sigma.\mathscr{I}_k(d\,|\,s)\sigma\backslash\delta.$$

Proof. Let $\sigma \in \Sigma$. Suppose $\mathscr{I}_k(d\,|\,s)\sigma = \theta\zeta$ with $\theta \in \Sigma^*$, $\zeta \in \{\varepsilon, \perp, \delta\}$. Then we have $[\langle s, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k} Z$ with $Z \in \{E, \Omega, \Delta\}$ appropriate. So $[\langle s, E\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k} Z\backslash\Delta$ by application of $\cdot\backslash\Delta$, and $[\langle s, E\rangle{:}\langle \varepsilon, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k+1} Z\backslash\Delta$ by application of $\cdot{:}\langle \varepsilon, \Delta\rangle$. Since

$$[\langle x, \Delta\rangle, \sigma] \rightarrow^{r}_{d,k+1} [\langle s, E\rangle{:}\langle \varepsilon, \Delta\rangle, \sigma]$$

we have $[\langle x, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k+1} Z\backslash\Delta$. By definition of $\cdot\backslash\Delta$ it follows that

$$\mathscr{I}_{k+1}(d\,|\,x)\sigma = \theta(\zeta\backslash\delta) = (\theta\zeta)\backslash\delta = \mathscr{I}_k(d\,|\,s)\sigma\backslash\delta. \qquad \square$$

The last lemma in this section in devoted to the compositionality of $\mathscr{I}_k$ with respect to sequential composition. For the introduction of a statement s in computations of the statement e, obtaining (parts of) computations of $e;s$ we define the operator $\cdot;s$. However, this only makes sense for frames having a generalized statement of at least one component, i.e., distinct from γ. (Note that stacks occurring in transition sequences starting from an initial stack, i.e., a stack of the format $[\langle s, \Delta\rangle, \sigma]$, have at most one occurrence of γ. Moreover, if γ occurs, then it does so in the top frame, as can be established by induction on the length of the computation.) Therefore, we first put

$$\varepsilon Stack' = \{[g_1, \sigma_1]{:}\ldots{:}[g_r, \sigma_r]{:}Z \in Stack' \mid r \in \mathbb{N}, g_i \neq \gamma, Z \in \{E, \Delta\}\},$$

$$\varepsilon Conf' = \varepsilon Stack' \cup \{\Omega\},$$

and for $s \in Stat$ we define $\cdot;s{:}\varepsilon Conf' \to \varepsilon Conf'$ by $\Omega;s = \Omega$, $E;s = E$, $\Delta;s = \Delta$, and if $F = [\langle s_1, D_1\rangle{:}\ldots{:}\langle s_r, D_r\rangle{:}\langle s_{r+1}, D_{r+1}\rangle, \sigma]$ then $(F{:}S);s = \bar{F}{:}(S;s)$ where

$$\bar{F} = [\langle s_1, \bar{D}_1\rangle{:}\ldots{:}\langle s_r, \bar{D}_r\rangle{:}\langle s_{r+1};s, \bar{D}_{r+1}\rangle, \sigma] \quad \text{and} \quad \bar{D}_i = D_i;s.$$

As before (but now for $C, C' \in \varepsilon Conf'$) we have that $C \twoheadrightarrow^{\theta}_{d,k} C'$ implies $C;s \twoheadrightarrow^{\theta}_{d,k} C';s$ as can be easily checked by inspection of the clauses of $\rightarrow_{d,k}$.

Lemma 5.12. *For $k \in \mathbb{N}$, $d \in Decl$, $e \in EStat$ and $s \in Stat$ we have*

$$\mathscr{I}_k(d\,|\,e;s) = \mathscr{I}_k(d\,|\,s) \circ \mathscr{I}_k(d\,|\,e).$$

Proof. Let $\sigma \in \Sigma$. To prove $\mathscr{I}_k(d\,|\,e;s)\sigma = \mathscr{I}_k(d\,|\,s)^{\wedge}(\mathscr{I}_k(d\,|\,e)\sigma)$, suppose $[\langle e, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k} Z$ with $\theta = \sigma_1..\sigma_r \in \Sigma^*$, $Z \in \{E, \Omega, \Delta\}$ and $[\langle s, \Delta\rangle, \sigma_i] \twoheadrightarrow^{\theta_i}_{d,k} Z_i$ with $\theta_i \in \Sigma^*$, $Z_i \in \{E, \Omega, \Delta\}$. We claim:

(i) Suppose $\forall i \in \{1,..,r\}$: $Z_i = E$. Then it holds that $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} Z$ with $\tau = \theta_1..\theta_r$.

(ii) Suppose $\exists j \in \{1,..,r\}$ such that $\forall i \in \{1,..,j-1\}$: $Z_i = E$ and $Z_j = \Omega$. Then it holds that $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} \Omega$ with $\tau = \theta_1..\theta_j$.

(iii) Suppose $\exists j \in \{1,..,r\}$ such that $\forall i \in \{1,..,j-1\}$: $Z_i = E$ and $Z_j = \Delta$. Then it holds that $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} \Delta$ with $\tau = \theta_1..\theta_j$.

We check (iii), leaving the other similar cases to the reader: Choose $S_0,.., S_r \in Stack'$ such that

$$S_0 = [\langle e, \Delta\rangle, \sigma],$$

$$S_{i-1} \twoheadrightarrow^{\varepsilon}_{d,k} [\gamma, \sigma_i]:S_i \rightarrow^{\sigma}_{d,k} S_i,$$

$$S_r \twoheadrightarrow^{\varepsilon}_{d,k} Z.$$

From $S_{i-1} \twoheadrightarrow^{\varepsilon}_{d,k}[\langle \varepsilon, \Delta\rangle, \sigma_i]:S_i$ and $S_r \twoheadrightarrow^{\varepsilon}_{d,k} Z$ in $\varepsilon Conf'$ we derive

$$S_{i-1};s \twoheadrightarrow^{\varepsilon}_{d,k}[\langle s, \Delta\rangle, \sigma_i]:(S_i;s), \qquad S_r;s \twoheadrightarrow^{\varepsilon}_{d,k} Z, \quad Z \in \{E, \Omega, \Delta\}.$$

Since $\forall i \in \{1,.., j-1\}:[\langle s, \Delta\rangle, \sigma_i] \twoheadrightarrow^{\theta_i}_{d,k} E$ and $[\langle s, \Delta\rangle, \sigma_j] \twoheadrightarrow^{\theta_j}_{d,k} \Delta$ we have

$$\forall i \in \{1,.., j-1\}: \quad [\langle s, \Delta\rangle, \sigma_i]:(S_i;s) \twoheadrightarrow^{\theta_i}_{d,k} S_i;s,$$

$$[\langle s, \Delta\rangle, \sigma_j]:(S_j;s) \twoheadrightarrow^{\theta_j}_{d,k} \Delta$$

by insertion of $S_i;s$, $i \in \{1,.., j-1\}$, and $S_j;s$, respectively. So

$$[\langle e;s, \Delta\rangle, \sigma] = S_0;s \twoheadrightarrow^{\varepsilon}_{d,k} [\langle s, \Delta\rangle, \sigma_1]:(S_1;s) \twoheadrightarrow^{\theta_1}_{d,k} S_1;s$$

$$\twoheadrightarrow_{d,k} \cdots \twoheadrightarrow_{d,k} [\langle s, \Delta\rangle, \sigma_j]:(S_j;s) \twoheadrightarrow^{\theta_j}_{d,k} \Delta.$$

Conclusion: $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} \Delta$ with $\tau = \theta_1..\theta_j$.

Now assume $\mathscr{I}_k(d\,|\,s)\sigma = \theta\zeta$ with $\theta = \sigma_1..\sigma_r \in \Sigma^*$, $\zeta \in \{\varepsilon, \perp, \delta\}$ and $\mathscr{I}_k(d\,|\,s)\sigma_i = \theta_i\zeta_i$ with $\theta_i \in \Sigma^*$, $\xi_i \in \{\varepsilon, \perp, \delta\}$. Then we have $[\langle e, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta}_{d,k} Z$ and $[\langle s, \Delta\rangle, \sigma] \twoheadrightarrow^{\theta_i}_{d,k} Z_i$ with Z and Z_i corresponding to ζ and ζ_i, respectively. If $Z_i = E$ for all $i \in \{1,.., r\}$, then $\zeta_i = \varepsilon$, $i \in \{1,.., r\}$, and $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} E$ with $\tau = \theta_1..\theta_r$. So

$$\mathscr{I}_k(d\,|\,e;s)\sigma = \theta_1..\theta_j\perp = \theta_1\zeta_1 \cdots \theta_r\zeta_r\cdot\zeta$$

$$= \mathscr{I}_k(d\,|\,s)\hat{}(\sigma_1..\sigma_r\zeta) = \mathscr{I}_k(d\,|\,s)\hat{}(\mathscr{I}_k(d\,|\,e)\sigma).$$

If $Z_i = E$ for all $i \in \{1,.., j-1\}$ and $Z_j = \Omega$ for some $j \in \{1,.., r\}$, then $\zeta_i = \varepsilon$, $i \in \{1,.., j-1\}$, $\zeta_j = \perp$ and $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} \Omega$ with $\tau = \theta_1..\theta_j$. So

$$\mathscr{I}_k(d\,|\,e;s)\sigma = \theta_1..\theta_j\perp$$

$$= \theta_1\zeta_1 \cdots \theta_j\zeta_j \cdots \theta_r\zeta_r\cdot\zeta \quad \text{since } \zeta_j = \perp$$

$$= \mathscr{I}_k(d\,|\,s)\hat{}(\sigma_1..\sigma_r\zeta)$$

$$= \mathscr{I}_k(d\,|\,s)\hat{}(\mathscr{I}_k(d\,|\,e)\sigma).$$

If $Z_i = E$ for all $i \in \{1,.., j-1\}$ and $Z_j = \Delta$ for some $j \in \{1,.., r\}$, then $\zeta_i = \varepsilon$, $i \in \{1,.., j-1\}$, $\zeta_j = \delta$ and $[\langle e;s, \Delta\rangle, \sigma] \twoheadrightarrow^{\tau}_{d,k} \Delta$ with $\tau = \theta_1..\theta_j$. So

$$\mathscr{I}_k(d\,|\,e;s)\sigma = \theta_1..\theta_j\delta$$

$$= \theta_1\zeta_1 \cdots \theta_j\zeta_j \cdots \theta_r\zeta_r\cdot\zeta \quad \text{since } \zeta_j = \delta$$

$$= \mathscr{I}_k(d\,|\,s)\hat{}(\sigma_1..\sigma_r\zeta)$$

$$= \mathscr{I}_k(d\,|\,s)\hat{}(\mathscr{I}_k(d\,|\,e)\sigma). \qquad \square$$

Having gathered enough compositionality results we are now in the position to relate the restricted intermediate and denotational semantics.

Theorem 5.13. *For all $k \in \mathbb{N}$ we have $\mathscr{D}_k = \mathscr{I}_k$.*

Proof. To prove for all d, k and s: $\mathscr{D}_k(d \mid s) = \mathscr{I}_k(d \mid s)$. Induction on k and s. The cases "$\langle k, \varepsilon\rangle$", "$\langle k, \textbf{\textit{fail}}\rangle$", "$\langle k, a\rangle$", "$\langle k, !\rangle$" and "$\langle 0, x\rangle$" are straightforward. Consider "$\langle k+1, x\rangle$". Suppose $x \leftarrow s \in d$.

$$\begin{aligned}\mathscr{D}_{k+1}(d \mid x) &= [\![x]\!]_s \eta_{d,k+1} = \eta_{d,k+1} x = \Phi d \eta_{d,k} x \\ &= \lambda\sigma.[\![s]\!]_s \eta_{d,k} \sigma \backslash \delta = \lambda\sigma.\mathscr{I}_k(d \mid s)\sigma \backslash \delta \\ &= \mathscr{I}_{k+1}(d \mid x) \quad \text{by Lemma 5.11.}\end{aligned}$$

For "$\langle k, s_1 \textbf{ or } s_2\rangle$" we have

$$\begin{aligned}\mathscr{D}_k(d \mid s_1 \textbf{ or } s_2) &= [\![s_1 \textbf{ or } s_2]\!]_s \eta_{d,k} = [\![s_1]\!]_s \eta_{d,k} \cdot [\![s_2]\!]_s \eta_{d,k} \\ &= [\![s_1]\!]_s \eta_{d,k} \cdot [\![s_2]\!]_s \eta_{d,k} = \mathscr{I}_k(d \mid s_1) \cdot \mathscr{I}_k(d \mid s_2) \\ &= \mathscr{I}_k(d \mid s_1 \textbf{ or } s_2) \quad \text{by Lemma 5.10.}\end{aligned}$$

For "$\langle k, e;s\rangle$" we have

$$\begin{aligned}\mathscr{D}_k(d \mid e;s) &= [\![e;s]\!]_s \eta_{d,k} = [\![s]\!]_s \eta_{d,k} \circ [\![e]\!]_e \eta_{d,k} \\ &= \mathscr{I}_k(d \mid s) \circ \mathscr{I}_k(d \mid e) = \mathscr{I}_k(d \mid e;s) \quad \text{by Lemma 5.12.}\end{aligned}$$ □

By now all the ingredients for an equivalence proof for $\mathscr{O}$ and $\mathscr{D}$ are available, viz. Definition 5.3 defining the intermediate semantics, Theorem 5.7 that represents $\mathscr{I}$ as a chain, Theorem 5.13 that relates $\mathscr{I}_k$ and $\mathscr{D}_k$, Lemma 4.2 which states the continuity of the statement evaluator and helps going from $\mathscr{D}_k$ to $\mathscr{D}$ and, finally, Theorem 5.4 relating the operational and intermediate semantics.

Theorem 5.14. $\mathscr{O} = \mathscr{D} \backslash \delta$.

Proof. Let $d \in \mathit{Decl}$, $s \in \mathit{Stat}$. Then

$$\begin{aligned}\mathscr{I}(d \mid s) &= \mathit{lub}_k(\mathscr{I}_k(d \mid s)) = \mathit{lub}_k(\mathscr{D}_k(d \mid s)) \\ &= \mathit{lub}_k([\![s]\!]_s \eta_{d,k}) = [\![s]\!]_s(\mathit{lub}_k \eta_{d,k}) = [\![s]\!]_s \mathit{lfp}(\Phi d) = \mathscr{D}(d \mid s).\end{aligned}$$

So $\mathscr{O} = \mathscr{I} \backslash \delta = \mathscr{D} \backslash \delta$. □

6. Interpretation of $\mathscr{B}$ into PROLOG

In the previous sections we studied comparative semantics for the uniform or abstract language $\mathscr{B}$. $\mathscr{B}$ is called uniform for its actions. The effect of an action in

a state is given by an arbitrary chosen action interpretation. This now serves as a handle for the interpretation of $\mathcal{B}$ as the core of the programming language PROLOG, viz. a left-most depth-first implementation of Horn clause logic with backtracking and cut. By choosing an appropriate interpretation of actions, i.e., fixing a suitable function I, we change our point of view to logic programming. We also obtain an operational and denotational semantics for PROLOG as refinements of the operational and denotational semantics for $\mathcal{B}$ developed in Sections 3 and 4, respectively. Finally, it will be obvious how to carry over the equivalence proof for $\mathcal{B}$ to the situation of PROLOG. Thus presently obtaining a relationship between the operational and denotational semantics almost for free.

Since we take the language $\mathcal{B}$ as a starting point for describing PROLOG procedure names are not allowed to have more than one procedure body. Hence, we confine to PROLOG programs wherein the clauses have pairwise different head predicates only. This is no essential restriction in the presence of the ***or***-construct and explicit unification actions, i.e., goals of the format $t_1 = t_2$. (One can use a so-called homogeneous form for clauses, as in [28] and ***or*** together clauses with the same head predicate.) So we arrive at unifications as our collection of atomic actions, atoms instead of procedure names, while we shall write & instead of the sequencing constructor ;. The precise reformulation of $\mathcal{B}$ is given in the next definition.

Definition 6.1. Let $\mathcal{F}$ be a collection of function symbols, $\mathcal{V}$ a collection of variables and $\mathcal{R}$ a collection of predicate letters. Let *Term* denote the collection of terms generated by $\mathcal{F}$ over $\mathcal{V}$. Define the set of unifications

$$Unif = \{t_1 = t_2 \mid t_i \in Term\},$$

the set of atoms

$$Atom = \{R(t_1,.., t_k) \mid R \in \mathcal{R} \text{ of arity } k,\ t_i \in Term\},$$

the set of atomic goals

$$AGoal = \{t_1 = t_2, \textbf{\textit{fail}}, !, G_1 \textbf{ or } G_2, R(t_1,.., t_k) \mid t_i \in Term,\ G_i \in Goal,\ R \in \mathcal{R} \text{ of arity } k\},$$

the set of goals

$$Goal = \{A_1 \& .. \& A_r \mid r \in \mathbb{N},\ A_i \in AGoal\}$$

with ***true*** $\in Goal$ denoting the empty goal, the set of PROLOG programs

$$Prog = \{A_1 \leftarrow G_1 :...: A_r \leftarrow G_r \mid r \in \mathbb{N},\ A_i = R_i(\vec{t}_i) \in AGoal,\ i \neq j \Rightarrow R_i \neq R_j,\ G_i \in Goal\}.$$

Define

$$\text{PROLOG} = \{P|G \mid P \in Prog,\ G \in Goal\}.$$

Next we develop an operational semantics for PROLOG along the lines of Section 3. In order to obtain a most general answer substitution one has to avoid clashes of logical variables. Here we try to resolve an atom against a program clause with variables that are fresh with respect to the computation so far. We can achieve this by having infinite supply of copies of the class of variables and tagging every goal with an index that it should be renamed with. (This is in fact structure sharing.) In a global counter we keep track of the number of the class of variables to rename with.

Definition 6.2. Let $Term'$ be the set of terms generated by $\mathscr{F}$ over $\mathscr{V} \times \mathbb{N}$ and Σ be the collection of substitutions over $Term'$, i.e.,

$$\Sigma = \{\sigma : Term' \to Term' \mid \sigma \text{ homomorphic}\}.$$

The set $GGoal$ of generalized goals is defined by

$$GGoal = \{\langle G_1, D_1, m_1\rangle :..: \langle G_r, D_r, m_r\rangle \mid r \in \mathbb{N},\ G_i \in Goal,\ D_i \in Stack,\ i \leqslant j \Rightarrow D_i \geqslant_{ss} D_j,\ m_i \in \mathbb{N}\},$$

the set of frames

$$Frame = \{[g, \sigma, n] \mid g \in GGoal,\ \sigma \in \Sigma,\ n \in \mathbb{N}\},$$

the set of stacks

$$Stack = \{F_1 :..: F_r \mid r \in \mathbb{N},\ F_i = [\langle G_1, D_1, m_1\rangle :..: \langle G_q, D_q, m_q\rangle, \sigma, n] \in Frame \text{ such that } F_{i+1} :..: F_r \geqslant_{ss} D_j\}$$

and the set of configurations

$$Conf = Stack \cup \{\Omega\}.$$

The set of labels is defined by $\Lambda = \Sigma \cup \{\varepsilon\}$.

The transition system underlying the operational semantics is a straightforward modification of Definition 3.3. We use a black box unification algorithm *mgu* that yields a most general unifier for two atoms or terms if one exists, and is undefined otherwise [17, 19]. We fix an action interpretation $I : Unif \to \mathbb{N} \to B \to_{part} B$ with $I(t_1 = t_2)m(\sigma, n) = (\sigma\theta, n)$ if $\theta = mgu(t_1^{(m)}\sigma, t_2^{(m)}\sigma)$ exists, and undefined otherwise. Here B denotes the collection of bindings defined by $B = \Sigma \times \mathbb{N}$. A binding $\beta = (\sigma, n)$ holds the current substitution and the value for the global counter of renaming indices.

Execution of actions $t_1 = t_2$ and procedure calls $R(t_1,.., t_k)$ involve unification. The effect of the execution of an action $t_1 = t_2$ in state σ is the update $\sigma\theta$, i.e., composition of substitutions, of σ with respect to the most general unifier θ of t_1 and t_2 in state σ (and appropriately renamed).

Slightly more deviating is procedure handling, since one has to unify first the call and the head of the particular clause successfully before body replacement can take place. A call is operationally described as follows. Consider a call, i.e., atom,

$R(t_1,.., t_k)$. First the concerning procedure definition, i.e. clause, is looked up in the declaration, i.e. PROLOG program. Say this is $R(\bar{t}_1,.., \bar{t}_k) \leftarrow \bar{G}$. (If R is not declared in the program a failure occurs and the frame is popped of the stack.) Next we try to unify $R(t_1,.., t_k)$ and $R(\bar{t}_1,.., \bar{t}_k)$ (considering renaming and the current substitution). If this is possible, i.e., a most general unifier exists, we replace the call by the procedure body, i.e., body of the program clause, extended with dump stack and renaming index, and change the state and global counter according to the side effect, i.e., the result of *mgu*, initiated by the call. We refer the reader to the nice tutorial of [22] for a discussion of unification in logic programming vs. parameter passing and value return in imperative languages.

Definition 6.3. Let $P \in Prog$. P induces a deterministic transition system $\rightarrow_P$ with as transition relation the smallest subset of $Conf \times \Lambda \times Conf$ such that

(i) $[\gamma, \sigma, n]:S \rightarrow_P^{\sigma} S$;

(ii) $[\langle \textbf{\textit{true}}, D, m\rangle:g, \sigma, n]:S \rightarrow_P [g, \sigma, n]:S$;

(iii) $[\langle t_1 = t_2 \& G, D, m\rangle:g, \sigma, n]:S \rightarrow_P [\langle G, D, m\rangle:g, \sigma\theta, n]:S$,
if $\theta = mgu(t_1^{(m)}\sigma, t_2^{(m)}\sigma)$ exists,
$[\langle t_1 = t_2 \& G, D, m\rangle:g, \sigma, n]:S \rightarrow_P S$, otherwise;

(iv) $[\langle \textbf{\textit{fail}} \ \& \ G, D, m\rangle:g, \sigma, n]:S \rightarrow_P S$;

(v) $[\langle ! \ \& \ G, D, m\rangle:g, \sigma, n]:S \rightarrow_P [\langle G, D, m\rangle:g, \sigma, n]:D$;

(vi) $[\langle R(t_1,.., t_k) \ \& \ G, D, m\rangle:g, \sigma, n]:S$
$\rightarrow_P [\langle \bar{G}, S, n\rangle:\langle G, D, m\rangle:g, \sigma\theta, n+1]:S$,
if $R(\bar{t}_1,.., \bar{t}_k) \leftarrow \bar{G} \in P$ and
$\theta = mgu(R(t_1^{(m)},.., t_k^{(m)})\sigma, R(\bar{t}_1^{(n)},.., \bar{t}_k^{(n)}))$ exists,
$[\langle R(t_1,.., t_k) \ \& \ G, D, m\rangle:g, \sigma, n]:S \rightarrow_P S$, otherwise;

(vii) $[\langle (G_1 \textbf{ \textit{or} } G_2) \ \& \ G, D, m\rangle:g, \sigma, n]:S \rightarrow F_1:F_2:S$,
where $F_i = [\langle G_i \ \& \ G, D, m\rangle:g, \sigma, n]$.

In the above definition we denote by $t^{(m)}$ the term in $Term'$ obtained by renaming in t variables in $\mathcal{V}$ into the corresponding variables in $\mathcal{V} \times \{m\}$. We use suffix notation for the application and composition of substitutions.

The operational semantics is defined similar to Definition 3.4. Here, in the context of logic programming, we choose to fix the start state, viz. the identity substitution σ_{id}. The renaming index is set to 1 having used 0 for the top level goal already.

Definition 6.4. The operational PROLOG semantics $\mathcal{O}: \text{PROLOG} \rightarrow \Sigma^{st}$ is defined by

$$\mathcal{O}(P \mid G) = \alpha_P([\langle G, E, 0\rangle, \sigma_{id}, 1])$$

where $\alpha_P: Conf \rightarrow \Sigma^{st}$ is the valuation associated with the transition system induced by P.

The denotational semantics for PROLOG with cut is a straightforward modification of Definition 4.1 taking into account renaming indices and the tratement of procedures. Note that instead of streams over states we presently deliver streams over bindings, i.e., elements of B_δ^{st}. The use of bindings instead of states is necessitated by the conjunction (i.e., sequential composition) $A \,\&\, G$. Here information has to be passed from A to G concerning the classes of variables used in the computation of A.

Definition 6.5. Let $Env = Atom \to \mathbb{N} \to B \to B_\delta^{st}$.

(i) $[\![\cdot]\!]_A : AGoal \to Env \to \mathbb{N} \to B \to B_\delta^{st}$,
$[\![t_1 = t_2]\!]_A \eta m = I(t_1 = t_2)m$,
$[\![\textit{fail}]\!]_A \eta m = fail$,
$[\![!]\!]_A \eta m = cut$,
$[\![G_1 \textbf{ or } G_2]\!]_A \eta m = ([\![G_1]\!]_G \eta m) \cdot ([\![G_2]\!]_G \eta m)$,
$[\![R(\vec{t})]\!]_A \eta m = \eta\{R(\vec{t})\}m$;

(ii) $[\![\cdot]\!]_G : Goal \to Env \to \mathbb{N} \to B \to B_\delta^{st}$,
$[\![\textit{true}]\!]_G \eta m = id$,
$[\![A \,\&\, G]\!]_G \eta m = ([\![G]\!]_G \eta m) \circ ([\![A]\!]_A \eta m)$;

(iii) $\Phi : Prog \to Env \to Env$,
$\Phi P \eta\{R(\vec{t})\}m(\sigma, n) = [\![G_0]\!]_G \eta n(\sigma\theta, n+1) \setminus \delta$,
if $R(\vec{t}_0) \leftarrow G_0 \in P$ and $\theta = mgu(R(\vec{t}^{(m)})\sigma, R(\vec{t}_0^{(n)}))$ exists,
$\Phi P \eta\{R(\vec{t})\}m(\sigma, n) = \varepsilon$ otherwise;

(iv) $[\![\cdot]\!]_{\text{Prolog}} : \text{PROLOG} \to B_\delta^{st}$,
$[\![P \mid G]\!]_{\text{Prolog}} = [\![G]\!]_G \eta_P 0(\sigma_{id}, 1)$,
where η_P is the least fixed point of $\Phi(P)$.

It is a matter of routine to obtain the equivalence of the operational and denotational semantics for PROLOG along the lines of Section 5.

Theorem 6.6. *Define the projection* $\pi : B \to \Sigma$ *by* $\pi(\sigma, n) = \sigma$. *Then it holds that* $\mathcal{O} = \pi \circ (\mathcal{D} \setminus \delta)$.

7. Concluding remarks

In this paper we have presented equivalent operational and denotational models for PROLOG with cut. First an abstract language embodying the *flow of control* component including ! was studied. For this language we have given a transition-system-based operational semantics and a direct denotational semantics. Moreover, we have related this models by intervention of an intermediate semantics and comparing appropriate approximations. At the level of this approximations the transition sequences are finite. This gave us a convenient tool for deriving the compositionality of the restrictions of the intermediate semantics. Secondly we

interpreted and refined the abstract language by adding the *logic* component of PROLOG. The semantical definitions as well as their relationship induced by this change of view provided us with equivalent operational and denotational models for PROLOG with cut itself.

Since our operational and denotational semantics for PROLOG are equal modulo hiding of the cut marker it remains to address the question of full abstractness of our semantics (cf. [1, 17, 18]). Other future work concerns the applicability of our use of approximations in comparing e.g. concurrency semantics.

Acknowledgement

Thanks are due to Jaco de Bakker, Frank de Boer, Arie de Bruin, Joost Kok, John-Jules Meyer and Jan Rutten for the fruitful discussions on preliminary versions of this paper in the seminar of the Amsterdam Concurrency Group.

References

[1] K.R. Apt and M.H. van Emden, Contributions to the theory of logic programming, *J. ACM* **29** (1982) 841-862.

[2] B. Arbab and D.M. Berry, Operational and denotational semantics of Prolog, *J. Logic Programming* **4** (1987) 309-329.

[3] M. Badinet, Proving termination properties of PROLOG programs: A semantic approach, in: *Proceedings LICS'88*, Edinburgh, Scotland (1988) 336-347.

[4] J.C.M. Baeten and W.P. Weijland, Semantics for Prolog via term rewrite systems, in: S. Keplan and J.-P. Jouannaud, eds., *Proceedings Conditional Term Rewriting Systems*, Lecture Notes in Computer Science **308** (Springer, Berlin, 1987).

[5] M. Broy, Theory for nondeterminism, parallelism, communication and concurrency, *Theoret. Comput. Sci.* **45** (1986) 1-62.

[6] J.W. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall International, London, 1980).

[7] J.W. de Bakker, Comparative semantics for flow of control in logic programming without logic, Rept. CS-R8840, Centre for Mathematics and Computer Science, Amsterdam (1988), also *Inform. and Comput.* (to appear).

[8] J.W. de Bakker and J.N. Kok, Uniform abstraction, atomicity and contractions in the comparative semantics of concurrent Prolog, in: *Proceedings FGCS'88*, Tokyo (1988) 347-355.

[9] J.W. de Bakker, J.N. Kok, J.-J.C. Meyer, E.-R. Olderog and J.I. Zucker, Contrasting themes in the semantics of imperative concurrency, in: J.W. de Bakker, W.P. de Roever and G. Rozenberg, eds., *Current Trends in Concurrency: Overviews and Tutorials*, Lecture Notes in Computer Science **224** (Springer, Berlin, 1986) 51-121.

[10] J.W. de Bakker and J.-J.C. Meyer, Metric semantics for concurrency, *BIT* **28** (1988) 504-529.

[11] J.W. de Bakker, J.-J.C. Meyer, E.-R. Olderog and J.I. Zucker, Transition systems, metric spaces and ready sets in the semantics for uniform concurrency, *J. Comput. Syst. Sci.* **36** (1988) 158-224.

[12] S.K. Debray and P. Mishra, Denotational and operational Semantics for Prolog, *J. Logic Programming* **5** (1988) 61-91.

[13] A. de Bruin and E.P. de Vink, Continuation semantics for Prolog with cut, in: *Proceedings Computing Science in the Netherlands*, Utrecht (1988) 295-314.

[14] P. Deransart and G. Ferrand, An operational formal definition of PROLOG, Rapport de Recherche 598, INRIA (1986).

[15] E.P. de Vink, Equivalence of an operational and a denotational semantics for a Prolog-like language with cut, Rept. IR-151, Free University, Amsterdam (1988).
[16] M. Fitting, A deterministic PROLOG fixpoint semantics, *J. Logic Programming* **2** (1985) 111–118.
[17] G. Frandsen, A denotational semantics for logic programming, DAIMI PB-201, Aarhus University, Aarhus (1985).
[18] R. Gerth, M. Codish, Y. Lichtenstein and E. Shapiro, Fully abstract denotational semantics for Concurrent Prolog, in: *Proceedings LICS'88*, Edinburgh, Scotland (1988) 320–335.
[19] N.D. Jones and A. Mycroft, Stepwise development of operational and denotational semantics for Prolog, in: *Proceedings Symposium on Logic Programming*, Atlantic City, NJ (1984) 281–288.
[20] J.N. Kok and J.J.M.M. Rutten, Contractions in comparing concurrency semantics, in: T. Lepistö and A. Salomaa, eds., *Proceedings ICALP'88*, Lecture Notes in Computer Science **317** (Springer, Berlin, 1988) 317–332.
[21] R. Kowalski, Algorithm = Logic + Control, *Comm. ACM* **22** (1979) 424–436.
[22] G. Levi, Logi programming: The foundations, the approach and the role of concurrency, in: J.W. de Bakker, W.P. de Roever and G. Rozenberg, eds., *Current Trends in Concurrency: Overviews and Tutorials*, Lecture Notes in Computer Science **224** (Springer, Berlin, 1986) 396–441.
[23] J.W. Lloyd, *Foundations of Logic Programming* (Springer, Berlin, 1984).
[24] J.-J.C. Meyer, Programming calculi based on fixed point transformations: Semantics and applications, Dissertation, Free University, Amsterdam (1985).
[25] J.-J.C. Meyer and E.P. de Vink, Step semantics for 'True' concurrency with recursion, *Distributed Comput.* **3** (1989) 130–145.
[26] G.D. Plotkin, A structural approach to operational semantics, DAIMI FN-19, Aarhus University, Aarhus (1981).
[27] J.E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory* (MIT Press, Cambridge, MA, 1977).
[28] M.H. van Emden and K. Yukawa, Logic programming with equations, *J. Logic Programming* **4** (1987) 265–288.