Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.W. de Bakker

Comparative semantics for flow of control in
logic programming without logic

* Computer Science/Department of Software Technology Report CS-R8840 October

---

Centrum voor Wiskunde en Informatica
Amsterdam

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.).

Comparative Semantics for Flow of Control in Logic Programming without Logic

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

We study semantic issues concerning control flow notions in logic programming languages by exploring a two-stage approach. The first stage considers solely uninterpreted (or schematic) elementary actions, rather than operations such as unification, substitution generation or refutation. Accordingly, logic is absent at this first stage. We provide a comparative survey of the semantics of a variety of control flow notions in (uninterpreted) logic programming languages including notions such as don't know versus don't care nondeterminism, the cut operator, and/or parallel logic programming, and the commit operator. In all cases considered, we develop operational and denotational models, and prove their equivalence. A central tool both in the definitions and in the equivalence proofs is Banach's theorem on (the uniqueness of) fixed points of contracting functions on complete metric spaces. The second stage of the approach proceeds by interpreting the elementary actions, first as arbitrary state transformations, and next by suitably instantiating the sets of states and of state transformations (and by articulating the way in which a logic program determines a set of recursive procedure declarations). The paper concentrates on the first stage. For the second stage, only a few hints are included. Furthermore, references to papers which supply details for the languages PROLOG, CONCURRENT PROLOG and GUARDED HORN CLAUSES are provided.

Keywords and Phrases: logic programming, operational semantics, denotational semantics, don't know nondeterminism, cut operator, and/or parallel logic programming, don't care nondeterminism, commit operator, fixed points of contracting functions, metric process theory, grain size of atomic actions.
1985 Mathematics Subject Classification: 68Q55, 68Q10, 68N15.
1987 Computing Reviews Categories: D.1.3, D.3.1, F.1.2, F.3.2.

1. Introduction

We report on the first stage of an investigation of the semantics of imperative concepts in logic programming. Logic programming being logic + control ([Kw]), one may expect to be able to profit from the large body of techniques and results in the semantic modelling of control flow gathered over the years. We shall, in fact, take a somewhat extreme position, and ignore in the analysis below *all* aspects having to do with logic. Rather, we shall provide a systematic treatment of a number of fundamental control flow concepts as encountered in logic programming on the basis of a model where the atomic steps are uninterpreted elementary actions. This constitutes a major abstraction step at two levels. Syntactically, we abstract from all structure in the atoms (using symbols from some alphabet rather than terms involving variables, functions or predicate symbols). Semantically, we abstract from any articulation in the basic computation steps, thus ignoring concepts such as unification, (SLD-) resolution or substitution generation. Does there remain anything interesting after this abstraction step? If yes, do the remnants shed any light on logic programming semantics? These two questions are addressed in our paper, and it is our aim to collect sufficient evidence that the answers to them are affirmative. More specifically, we want to argue that the semantic analysis of the collection of control flow concepts as provided below is justified for at least three reasons:

- It helps in clarifying basic properties of control flow phenomena. For example, we shall study versions of the cut operator, notions in and/or parallel programming such as don't know nondeterminism versus don't care nondeterminism and the commit operator, and it may be difficult

Report CS-R8840
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

to grasp these concepts in the presence of the full machinery of logic programming.

- We shall systematically provide *operational* and *denotational* models for the various example languages introduced below, and develop a uniform method to establish the equivalence of these semantics in all cases. We see as a main achievement the gathering of evidence that for such comparative semantics it is sufficient to work at the uninterpreted level. For both the operational and the denotational models, an interpretation towards the detailed level of logic programming may then be performed subsequently, if desired. The demonstration of the power of the uniform proof principle which turns out to be applicable in all cases studied, may be see as a subsidiary goal of our investigation.
- Altogether, we shall deal with six example languages, each embodying a small (and varying) collection of control flow concepts. Seemingly small variations in the language concepts require careful tuning of the semantic tools, sometimes involving substantial modification of the models employed. Thus, leaving the origin of the concepts aside for a moment, one may view our paper as a contribution to comparative control flow semantics in general. The confrontation of the (dis) similarities encountered throughout may provide an illuminating perspective on some of its fundamental issues. It should be added here that, from the methodological point of view, our (exclusive) use of metric methods may be seen as well as a distinguishing feature.

The answer to the second question — what is the relevance of all this for full logic programming semantics—awaits further work. Much will depend on the feasibility of obtaining this full semantics simply by interpreting the elementary actions as computational steps in the sense of the relevant version of the logic programming language, leaving the already available (abstract) control flow model intact. A substantial part of the detailed work in establishing this still has to be done. On the other hand, there are already a few case studies available which may be seen as providing support for our thesis. A promising first step is made in [Vi, BrV], where for a simple PROLOG-like language it is firstly shown how to add interpretations of elementary actions as (arbitrary) state transformations to the semantic model(s). This, in turn, allows a smooth transition towards a model incorporating essential elements of a declarative semantics for PROLOG: instead of the delivery of (sequences of) states, by suitably specializing them the semantic definitions are now geared to the delivery of (sequences of) *substitutions* (in the familiar sense of logic programming). A second paper which follows the approach indicated above is [BK]. This paper continues earlier work of Kok [Ko] reporting on a branching time model for Concurrent Prolog (abbreviated as CP, and stemming from [Sh1]), where the use of a branching time model is in particular motivated by CP's commit operator. In [BK], an intermediate language is introduced with arbitrary interpretations for its atomic actions, and operational and denotational models are developed for it. Next, by suitably choosing both the sets of atomic actions and of procedure variables, by choosing one particular interpretation function (involving the determination of most general unifiers), and by using the information in the CP program to infer the declarations for the procedure variables, an induced comparative semantics for CP is obtained. Finally, in the paper [BoKPR] the semantics of Guarded Horn Clauses (GHC) is studied both from the control flow and from the classical (declarative) point of view. More specifically, first an approach following the intermediate language technique — the same one as that used for CP in [BK] — is described. Next, a declarative semantics, in terms of familiar notions such as Herbrand base or immediate consequence operator, is developed which allows one to determine the success set of a GHC program. (In an appendix to the present paper, we shall present a brief sketch of the interpretation chosen for a rudimentary form of (and/or) parallel logic programming - based essentially on ideas of Kok from [BK] -, in order to illustrate the feasibility of obtaining logic programming with logic by suitably interpreting logicless languages.)

We shall now be somewhat more specific as to which control flow concepts will be investigated. In various groupings, we shall deal with the following notions

- elementary action
- (procedure declarations and) recursion

- failure
- sequential execution
- backtracking or don't know non-determinism
- cut (in two versions, to be called *absolute* and *relative* cut)
- parallel execution
- (don't care) non-determinism
- commit.

These notions are grouped into six languages, L_1 to L_6. Each has elementary actions, recursion and failure, and the precise distribution of the other concepts over the languages can be inferred from the syntax overview to be presented at the end of this introduction. Notable imperative concepts missing from the above list—taking our decision to start from uninterpreted elementary actions for granted—are synchronization and process creation. We have omitted them for no other reason than our wish not to overload the present paper. We plan to include these concepts which are indeed pervasive in many versions of parallel logic programming in a subsequent publication.

For each of the languages L_1 to L_6 we shall present both operational and denotational semantics. The operational semantics will be based on labelled transition systems ([Ke]), embedded in a syntax directed deductive system in the style of Plotkin's Structured Operational Semantics ([HP, Pl1,2]). The denotational models will be built on metric structures (as will be the way in which we infer operational meanings through the assembling of information in transition sequences). Partly, these structures will be of the *linear time* variety, i.e., they will consist of (nonempty closed) sets of finite or infinite sequences over some alphabet. Partly, we shall work with *branching time* domains. More precisely, the meaning of a statement will be a process (in the sense of [BZ]), i.e., an element of a mathematical domain which is obtained as solution of a domain equation to be solved using metric tools. Roughly, such a process is like a tree over the relevant alphabet of elementary actions, satisfying various additional properties (commutativity, absorption, closedness).

For the logic part of the semantics of logic programming we refer to the book [Ll] or to the comprehensive survey of APT [Ap]. A recent tutorial on and comparison of *parallel* logic programming languages is the paper by RINGWOOD [Ri]. Our interest in comparative logic programming semantics, using techniques which fit more in the imperative than in the logic tradition was originally raised by [JM]. Elsewhere, we have often used the term 'uniform' for uninterpreted or schematic languages (in general), e.g. in [BMO, BMOZ, BKMOZ], and the present investigation may also be seen as a semantic exploration of uniform versions of logic programming, with special emphasis on the comparative aspects. Other papers which address operational versus denotational semantics for PROLOG are [DM, ABe, VV, Vi, BrV]. We shall return to the latter two below. We already mentioned [BK] on semantic equivalence for Concurrent Prolog. In [GCLS], both operational and denotational semantics are presented for Flat Theoretical Concurrent Prolog (from [Sh2]). Whereas in [Ko, BK] the denotational models are based on processes as in [BZ], in [GCLS] the failure set model of [BHR] is applied. In addition, [GCLS] discusses full abstractness issues. A detailed analysis of operational semantics for (variations on) CP is provided in [Sa]. The papers such as [Ko, BK, JM, GCLS] should all be situated primarily in the tradition of imperative concurrency semantics, rather than pursuing the line of extending the declarative semantics approach of 'classical' logic programming in terms of (generalizations of) Herbrand universes. It is the latter approach which is followed in [LP2], where a detailed comparison is given of synchronization phenomena in a variety of parallel logic programming languages. The paper [LP1] concentrates in particular on the declarative semantics of CP's read - only variables. Related references include [FL, FLMP, FOM, Le].

For some time now, we have been utilizing metrically based semantic models, e.g. in [BZ, BBKM, BMOZ, BKMOZ, BM, ABKR]. An essential extension of the metric domain theory was provided in [AR]. An important advantage of the metric framework, compared with the usual order theoretic one, lies in the fact that many of the functions encountered in the semantic models are contracting and, hence, have unique fixed points (by Banach's theorem). This property may be exploited both in the semantic definitions proper (see [ABKR] for many examples), and in the derivation of semantic

equivalences. It is the latter technique, first described in [KR], which constitutes the powerful method already referred to, and which will be applied throughout our paper. (For further examples of the method see [BM].) Our model of the denotational semantics of backtracking is a uniform (schematic) version of a definition from [Br]. The operational semantics for the cut operator(s) were supplied by E.P. de Vink (personal communication).

We conclude this introduction with an outline of the contents of our paper. Section 2 contains some mathematical preliminaries, mainly devoted to the underlying metric framework. The overview of the remaining sections is best presented by listing the syntax of the languages studied in them. For each L_i, we define statements $s \in L_i$ which are to be executed with respect to a set of declarations D. Let A be the (possibly infinite) alphabet of elementary actions, with a ranging over A, and let *Pvar* be the alphabet of procedure variables, with x ranging over *Pvar*. The following operators will be encountered:

- sequential composition $s_1; s_2$
- don't know nondeterminism $s_1 [] s_2$
- absolute cut !
- relative cut !!
- parallel composition $s_1 \| s_2$
- don't care non-determinism $s_1 + s_2$
- commit $s_1 : s_2$

The languages, corresponding section headings, and respective syntactic definitions are summarized in

L_1 : sequential logic programming with backtracking
$s ::= a|x|$ **fail** $|s_1;s_2|s_1 [] s_2$

L_2 : sequential logic programming with backtracking and absolute cut
$s ::= a|x|$ **fail** $|s_1;s_2|s_1 [] s_2|$!

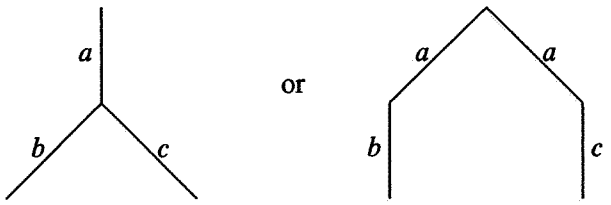
L_3 : sequential logic programming with backtracking and relative cut
$s ::= a|x|$ **fail** $|s_1;s_2|s_1 [] s_2|$!!

L_4 : (and/or) parallel logic programming: the linear time case
$s ::= a|x|$ **fail** $|s_1;s_2|s_1 \| s_2|s_1 + s_2$

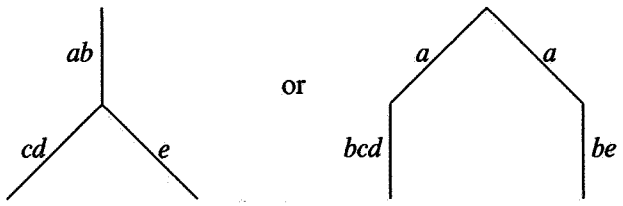
L_5 : (and/or) parallel logic programming with commit: the branching time case
$s ::= a|x|$ **fail** $|s_1:s_2|s_1 \| s_2|s_1 + s_2$

L_6: (and/or) parallel logic programming with commit: increasing the grain size
$s ::= a|x|$ **fail** $|s_1;s_2|s_1:s_2|s_1 \| s_2|s_1 + s_2$

For each language, a program in that language consists of a pair $<D|s>$, $s \in L_i$, $D \equiv <x_j \Leftarrow g_j>_j$, where g_j is a *guarded* statement from L_i—guarded here meaning that occurrences of *calls* (of some $x \in Pvar$) in g_j are preceded by some elementary action. Languages L_1 to L_3 are deterministic, and the main issue is how to model the backtracking and cut operators. Languages L_4 and L_5 are (very much stripped) versions of (and/or) parallel logic programming. The difference between these two consists in the transition from (normal) sequential composition (;) to commit (:). This induces a different failure behaviour which in turn leads to the definition of a *linear time* (LT) model for L_4 and a *branching time* (BT) model for L_5. An LT model (over an alphabet A) consists, as we saw earlier, of sets of sequences of elementary actions from A, whereas a BT model (also over A) consists of tree-like entities (with the already mentioned extra features). Maybe the technically most interesting issue of our paper is addressed in Section 8, where we combine the composition operations of sequential composition (;) and commit (:) into one language. Whereas for L_4 we encounter meanings such as, e.g., $\{ab, ac\}$ and, for L_5, processes such as



in L_6 we shall make use of meanings which have forms such as



Viewing the entities labelling the edges in the trees as the 'grains' of our model, we see that, in going from L_5 to L_6, we increase the grain size. We shall (in the context of L_6) interpret sequential composition as an operator which leads to larger atoms (or grains), and commit (just as for L_5) as an operator which induces branches is the trees. In a final section (Section 9) we introduce an alternative transition system for L_6, and show that this leads to the same operational (and denotational) semantics as that defined in Section 8. The appendix provides a brief sketch of a possible translation from a rudimentary logic programming language towards L_4.

2. Mathematical preliminaries

2.1. Notation

The notation $(x \in)X$ introduces the set X with typical element x ranging over X. For X a set, we denote with $\mathcal{P}(X)$ the power set of X, i.e., the collection of all subsets of X. $\mathcal{P}_\pi(X)$ denotes the collection of all subsets of X which have property π. A sequence $x_0, x_1, \cdots$ of elements of X is denoted by $(x_i)_{i=0}^{\infty}$ or, briefly, by $(x_i)_i$. The notation $f: X \to Y$ expresses that f is a function with domain X and range Y. We use the notation $f\{y/x\}$, with $x \in X$ and $y \in Y$, for a *variant* of f, i.e., for the function which is defined by

$$\begin{aligned} f\{y/x\}(x') &= y \quad , \text{ if } x = x' \\ &= f(x'), \text{ otherwise.} \end{aligned}$$

If $f: X \to X$ and $f(x) = x$, we call x *a fixed point* of f.

2.2. Metric spaces

Metric spaces are the mathematical structures in which we carry out our semantic work. We give only the facts most needed in this paper. For more details, the reader is referred to [Du, En].

Definition 2.1. A metric space is a pair (M, d) where M is any set and d is a mapping $M \times M \to [0,1]$ having 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. $\forall x, y, z \in M[d(x, y) \leqslant d(x, z) + d(z, y)]$.

The mapping d is called a *metric* or *distance.* In case d satisfies 3′ instead of 3:

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

we call d an *ultrametric.*

EXAMPLES.

1. Let A be an arbitrary set. The *discrete metric* on A is defined as follows: Let $x, y \in A$.

$$d(x, y) = 0 \text{ if } x = y$$
$$= 1 \text{ if } x \neq y.$$

2. Let A be an alphabet, and let $A^\infty = A^* \cup A^\omega$ denote the set of all finite and infinite words over A. Let, for $x \in A^\infty$, $x(n)$ denote the prefix of x of length n, in case *length* $(x) \geqslant n$, and x otherwise. We put

$$d(x, y) = 2^{-\sup\{n | x(n) = y(n)\}}$$

with the convention that $2^{-\infty} = 0$. Then (A^∞, d) is an ultrametric space.

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

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

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

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

$$\forall \epsilon > 0 \exists N \in N \forall n > N [d(x, x_n) < \epsilon].$$

We call the sequence $(x_i)_i$ *convergent* and write $x = \lim_i x_i$.

3. (M, d) is called *complete* whenever each Cauchy sequence in M converges to an element of M.

DEFINITION 2.3. Let (M_1, d_1) and (M_2, d_2) be metric spaces.

1. We say that (M_1, d_1) and (M_2, d_2) are *isometric* if there is a mapping $f : M_1 \to M_2$ such that
   (a) f is a bijection
   (b) $\forall x, y \in M_1 [d_2(f(x), f(y)) = d_1(x, y)]$.
   We then write $M_1 \cong M_2$. If we have a function f satisfying only condition (1b), we call it an isometric embedding.
2. Let $f : M_1 \to M_2$. We call f *continuous* whenever for each sequence $(x_i)_i$ with limit x in M_1, we have that $\lim_i f(x_i) = f(x)$
3. We call a function $f : M_1 \to M_2$ *contracting* if there exists a real number c with $0 \leqslant c < 1$ such that

$$\forall x, y \in M_1 [d_2(f(x), f(y)) \leqslant c.d_1(x, y)]$$

4. A function $f : M_1 \to M_2$ is called *non-distance-increasing* if

$$\forall x, y \in M_1 [d_2(f(x), f(y)) \leqslant d_1(x, y)]$$

We shall denote the set of all non-distance-increasing functions (*ndi*) from M_1 to M_2 by $M_1 \to^1 M_2$.

THEOREM 2.4.

1. *Let (M_1, d_1) and (M_2, d_2) be metric spaces, and let $f : M_1 \to M_2$ be a contracting function. Then f is continuous. The same holds for non-distance-increasing functions.*
2. *(Banach.)*
   Let (M, d) be a complete metric space. Each contracting function $f : M \to M$ has a unique fixed point which equals $\lim_i f^i(x_0)$ for arbitrary $x_0 \in M$. (Here $f^0(x_0) = x_0$ and $f^{i+1}(x_0) = f(f^i(x_0))$.)

It may be instructive to recall the proof of Theorem 2.4-2. Since f is contracting, the sequence $(f^i(x_0))_i$ is Cauchy sequence. By the completeness of (M, d), the limit $x = \lim_i f^i(x_0)$ exists. By the continuity of f (part 1), $f(x) = f(\lim_i f^i(x_0)) = \lim_i f^{i+1}(x_0) = x$. If, for some $y \in M$, $f(y) = y$ then, by the contractivity of f, $d(x, y) = d(f(x), f(y)) \leqslant c.d(x, y)$. Hence, since $c < 1$ we conclude that $d(x, y) = 0$, and $x = y$ follows.

DEFINITION 2.5. Let (M, d) be a metric space. A subset X of M is called *closed* whenever each converging sequence with elements in X has its limit in X.

DEFINITION 2.6. Let (M, d), (M_1, d_1), and (M_2, d_2) be (ultra) metric spaces.

1. We define a metric d_F on the set $M_1 \to M_2$ of all functions from M_1 to M_2 as follows: For every $f_1, f_2 \in M_1 \to M_2$ we put
$$d_F(f_1, f_2) = \sup_{x \in M_1} d_2(f_1(x), f_2(x))$$
2. We define a metric d_p on the Cartesion product $M_1 \times M_2$ by
$$d_p((x_1, y_1), (x_2, y_2)) = \max_{i \in \{1,2\}} d_i(x_i, y_i)$$
3. With $M_1 \sqcup M_2$ we denote the *disjoint union* of M_1 and M_2, which may be defined as $(\{1\} \times M_1) \cup (\{2\} \times M_2)$. We define a metric d_U on $M_1 \sqcup M_2$ as follows:
$$\begin{aligned} d_U(x, y) &= d_i(x, y) \text{ if } x, y \in \{i\} \times M_i \text{ for } i = 1 \text{ or } i = 2 \\ &= 1 \qquad \text{otherwise.} \end{aligned}$$
In the sequel we shall often write $M_1 \cup M_2$ instead of $M_1 \sqcup M_2$, implicitly assuming that M_1 and M_2 are already disjoint.
4. Let $P_c(M) = \{X | X \subseteq M, X \text{ closed}\}$. We define a metric d_H on $P_c(M)$, called the *Hausdorff distance,* as follows:
$$d_H(X, Y) = \max\{\sup_{x \in X} d(x, Y), \sup_{y \in Y} d(y, X)\}$$
where $d(x, Z) = \inf_{z \in Z} d(x, z)$ (here we use the convention that $\sup \varnothing = 0$ and $\inf \varnothing = 1$).

THEOREM 2.7. *Let (M, d), (M_1, d_1), (M_2, d_2), d_F, d_p, d_U, and d_H be as in Definition 2.6. In case d, d_1, d_2 are ultrametrics, so are $d_F, \cdots, d_H$. Now suppose in addition that (M, d), (M_1, d_1), and (M_2, d_2) are complete. We have that*

1. $(M_1 \to M_2, d_F)$ *(together with $(M_1 \to^1 M_2, d_F)$)*
2. $(M_1 \times M_2, d_p)$
3. $(M_1 \sqcup M_2, d_U)$
4. $(P_c(M), d_H)$.

are complete metric spaces. (Strictly speaking, for the completeness of $M_1 \to M_2$, the completeness of M_1 is not required.)

In the sequel we shall often write $M_1 \to M_2$, $M_1 \times M_2$, $M_1 \sqcup M_2$, $P_c(M)$, etc., when we mean the metric spaces with the metrics just defined.

The proofs of parts 1,2, and 3 of Theorem 2.7 are straightforward. Part 4 is more involved. It can be proved with the help of the following characterization of completeness of $(P_c(M), d_H)$:

THEOREM 2.8. *Let $(P_c(M), d_H)$ be as in Definition 2.6. Let $(X_i)_i$ be a Cauchy sequence in $P_c(M)$. We have*
$$\lim_i X_i = \{\lim_i x_i | x_i \in X_i, (x_i)_i \text{ a Cauchy sequence in } M\}$$

Theorem 2.8 is due to HAHN [Ha]. Proofs of Theorems 2.7 and 2.8 can be found, e.g., in [Du] or [En]. The proofs are also repeated in [BZ].

THEOREM 2.9 (Metric completion).
Let M be an arbitrary metric space. Then there exists a metric space $\overline{M}$ (called the completion of M)

together with an isometric embedding $i: M \to \overline{M}$ *such that*

1. $\overline{M}$ *is complete.*
2. *For every complete metric space* M' *and isometric embedding* $j: M \to M'$ *there exists a unique isometric embedding* $\bar{j}: \overline{M} \to M'$ *such that* $\bar{j} \circ i = j$.

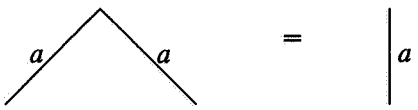
PROOF. Standard topology. □

2.3. Metric domain equations

We shall be interested in developing mathematically rigorous foundations for branching structures which are, in first approximation, nothing but (rooted) labelled trees (with labels from some set A) which satisfy three additional properties suggested by

1. commutativity

2. absorption



3. closedness (precise definition omitted)

We shall obtain the set of 'trees' satisfying these properties as the domain P of processes (with respect to A; this notion of process was introduced in [BZ]) satisfying the domain equation (or isometry)

$$P \cong \mathcal{P}_c(A \cup (A \times P)) \tag{2.1}$$

(Note that, for reasons of cardinality, (2.1) has no solution when we take *all* subsets rather than all closed subsets of $A \cup (A \times P)$.) More precisely, we want to solve (2.1) by determining P as a complete metric space (P, d) satisfying

$$(P, d) \cong \mathcal{P}_c(A \cup (A \times id_{\frac{1}{2}}(P, d))) \tag{2.2}$$

where the right-hand side is built up using the composite metrics of definition 2.6. In addition, we use the mapping $id_{\frac{1}{2}}$ where, for any real $c > 0$, $id_c(M, d) = (M, d_c)$, with $d_c(x, y) = c.d(x, y)$. (The use of the mapping $id_{\frac{1}{2}}$ is a technical—though essential—trick. Note that it affects only the metrics induced. Hence, (2.1) is a correct rendering of (2.2) when attention is restricted to the set components.) It has been shown in [BZ] how to solve equations such as (2.2): We define a sequence of complete metric spaces $((P_n, d_n))_{n=0}^{\infty}$, with $(P_0, d_0) = (\varnothing, d_0)$, d_0 arbitrary, and

$$P_{n+1} = \mathcal{P}(A \cup (A \times id_{\frac{1}{2}}(P_n, d_n)))$$

$$d_{n+1} = (\tilde{d}_n)_H.$$

Here $\tilde{d}_n$ is the metric determined (according to Definition 2.6) on $A \cup (A \times id_{\frac{1}{2}}(P_n, d_n))$, where we assume some given metric d_A on A. Next, we put $(P_\omega, d_\omega) = (\bigcup_n P_n, \bigcup_n d_n)$ (with the obvious interpretation of $\bigcup_n d_n$; note that $P_n \subseteq P_{n+1}$), and we define $(\overline{P}, \overline{d})$ as the *completion* (Theorem 2.9) of (P_ω, d_ω). Then we have.

THEOREM 2.10. $(\overline{P}, \overline{d})$ *is a complete metric space satisfying (2.2). If* d_A *is an ultrametric, then so is* $\overline{d}$.

PROOF. Essentially as in [BZ]. □

REMARKS.

1. The above explanation covers only one case out of a whole range of possible domain equations. In [AR], a category - theoretic treatment of the general case is described.
2. The reader who wonders about the connection between the process domain P and the models obtained through bisimulation from Milner's synchronization trees (or ACP's graph models) is referred to [BeK 1,2]. In a nutshell, in all relevant cases (assuming appropriate restrictions on the graph models) the domains considered are isomorphic.

3. SEQUENTIAL LOGIC PROGRAMMING WITH BACKTRACKING

The first language on our list, L_1, contains a combination of the features elementary action, recursion, failure, sequential composition and backtracking. It is intended as a uniform (uninterpreted) approximation to PROLOG, as yet without a cut operator (which will be added in Sections 4.5). We shall develop operational ($\mathcal{O}$) and denotational ($\mathcal{D}$) semantics for L_1. The two semantic models to be presented bring together certain previously proposed ideas from the literature in such a way that a smooth equivalence proof is made possible. The denotational model is a uniform variation of ideas in [Br], whereas the operational semantics for L_1 owes much to [Vi]. In [Vi], a denotational model is developed as well, though of the direct - no continuations - variety. An important technical difference between our work and that of [Vi] is that the latter is built on cpo structures (to be contrasted to our metric ones), and requires rather more effort to obtain the equivalence result. On the other hand, [Vi] handles arbitrary interpretations (rather than no interpretations), thus preparing the way for a transition towards actual PROLOG which consists in the choice of a specific interpretation: fixing the sets of elementary actions and procedure variables, interpreting the elementary actions (in terms of most general unifiers), determining the procedure declarations from the set of clauses in the PROLOG program, etc. This transition is described in detail in [BrVi], where also a continuation style denotational semantics for PROLOG with cut is developed, together with an equivalence proof in the cpo framework.

The equivalence proof we present below is an instance (many more follow in later sections) of a technique based on the idea that both $\mathcal{O}$ and $\mathcal{D}$ are fixed points of a contracting higher order operator (in a setting with an appropriate metric) and therefore coincide. This technique was first described in [KR] (for the metric case; see [AP] for an earlier order - theoretic argument). Various further examples can be found in [BM], all of which deal with programs with explicit (simultaneous) procedure declarations - as our languages L_1 to L_6 - rather than with programs where recursion appears through μ-constructs.

We begin with the definition of the syntax for L_1. Recall that a ranges over A, the set of elementary actions, and x over *Pvar*, the set of procedure variables. It will be convenient to assume that each program uses exactly the procedure variables in the initial segment $\mathcal{X}=\{x_1, \cdots, x_n\}$ of *Pvar*, for some $n \geqslant 0$.

DEFINITION 3.1. (Syntax)

a. (statements). The class $(s \in) L_1$ of *statements* is given by

$$s ::= a \mid x \mid \textbf{fail} \mid s_1 ; s_2 \mid s_1 \,[]\, s_2 \quad \text{with} \quad x \in \mathcal{X}$$

b. (guarded statements). The class $(g \in) L_1^g$ of *guarded statements* is given by

$$g ::= a \mid \textbf{fail} \mid g ; s \mid g_1 \,[]\, g_2$$

c. (declarations). The class $(D \in) \mathcal{D}ecl_1$ of *declarations* consists of n-tuples $D \equiv x_1 \Leftarrow g_1, \cdots, x_n \Leftarrow g_n$, or $<x_1 \Leftarrow g_i>_i$, for short, with $x_i \in \mathcal{X}$ and $g_i \in L_1^g$, $i = 1,2, \cdots, n$.

d. (programs). The class $(\sigma \in) \mathcal{P}rog$, of *programs* consists of pairs $\sigma \equiv <D|s>$, with $D \in Decl_1$ and $s \in L_1$.

EXAMPLE (assuming $a,b,c,d \in A$).

$$<x_1 \Leftarrow (a;x_2)[](b;x_3),\ x_2 \Leftarrow (c;x_1)[](d;\mathbf{fail}),\ x_3 \Leftarrow \mathbf{fail}|a;x_1;b>$$

REMARKS.

1. All g_i occurring in a declaration $D \equiv <x_i \Leftarrow g_i>_i$ are required to be guarded, i.e. occurrences of $x \in \mathcal{X}$ in g_i are to be proceeded by some g (which, by clause b, has to start with an elementary action). This requirement corresponds to the usual Greibach condition in language theory.
2. We have adopted the simultaneous declaration format for recursion rather than the μ-formalism which features (possibly nested) constructs such as, for example, $\mu x[(a;\mu y[(b;y)[]c];d)[]e]$. The simultaneous format is natural in the context of logic programming. Moreover, it allows a simpler derivation of the main semantic equivalence results presented below. (Certain additional inductive arguments applied in [KR] to deal with μ-constructs can now be avoided.)
3. Usually, we do not bother about parentheses around composite constructs. If one so wishes, parentheses may be added to avoid ambiguities.

We proceed with the definitions leading up to the *operational semantics* for $s \in L_1$ and $\sigma \in \mathcal{Prog}_1$. We introduce two auxiliary syntactic classes in

DEFINITION 3.2.

a. The class $(r \in)\mathcal{Rcon}_1$ of *success continuations* is defined by

$$r ::= E|(s;r)$$

b. The class $(t \in)\mathcal{Tcon}_1$ of *failure continuations* is defined by

$$t ::= \Delta|(r:t)$$

Here E and Δ are new symbols, and the parentheses around $(s;r)$ and $(r:t)$ will be omitted when no confusion is expected.

The semantic universe (both for operational and denotational semantics) for L_1 is quite simple. Let δ be a new symbol not in A, the intended meaning of which is to model failure. We define the semantic domain $(v,w \in)R$ in

DEFINITION 3.3. $R = A^* \cup A^\omega \cup A^*.\{\delta\}$. In other words, the elements of R (which will serve as meanings of statements or programs) are either finite sequences over A, possibly empty (ϵ) and possibly ending with δ,or infinite sequences over A. By Subsection 2.2, we can introduce a distance d on R which turns it into a complete ultrametric space (in the definition of d, δ plays the same role as the elements of A).

We now give the definitions of the operational semantics for L_1 and $\mathcal{Prog}_1$. They are based on *transition systems* (as in [HP, Pl1, Pl2]). Here, a transition is a fourtuple in $\mathcal{Tcon}_1 \times A \times Decl_1 \times \mathcal{Tcon}_1$, written in the notation

$$t \overset{a}{\rightarrow}_D t' \tag{3.1}$$

We present a *formal transition system* T_1 which consists of *axioms* (in the form as in (3.1)) or *rules*, in the form

$$\frac{t_1 \overset{a}{\rightarrow}_D t'}{t_2 \overset{a}{\rightarrow}_D t''}.$$

Transitions which are given as axioms *hold* by definition. Moreover, a transition which is the consequence of a rule holds in T_1 whenever it can be established that, according to T_1, its premise holds (or, in Section 9, premises hold). We shall employ below notational abbreviations for the rules such as (dropping the a and D in $\overset{a}{\rightarrow}_D$ for convenience)

$$\frac{t_1 \to t_2 | t_3}{t_1' \to t_2' | t_3'} \quad \text{as shorthand for} \quad \frac{t_1 \to t_2}{t_1' \to t_2'} \quad \text{and} \quad \frac{t_1 \to t_2}{t_1' \to t_3'}$$

and

$$\frac{\begin{matrix} t_1 \to t_2 \\ t_3 \to t_4 \\ t_5 \to t_6 \end{matrix}}{} \quad \text{as shorthand for} \quad \frac{t_1 \to t_2}{t_3 \to t_4} \quad \text{and} \quad \frac{t_1 \to t_2}{t_5 \to t_6}.$$

DEFINITION 3.4 (transition system T_1).

$$(a;r)\colon t \xrightarrow{a}_D (r\colon t) \tag{Elem}$$

$$\frac{(g;r)\colon t \xrightarrow{a}_D \tilde{t}}{(x;r)\colon t \xrightarrow{a}_D \tilde{t}}, \quad x \Leftarrow g \text{ in } D \tag{Rec}$$

$$\frac{t \xrightarrow{a}_D \tilde{t}}{(\mathbf{fail};r)\colon t \xrightarrow{a}_D \tilde{t}} \tag{Fail}$$

$$\frac{s_1;(s_2;r)\colon t \xrightarrow{a}_D \tilde{t}}{(s_1;s_2);r\colon t \xrightarrow{a}_D \tilde{t}} \tag{Seq Comp}$$

$$\frac{(s_1;r)\colon ((s_2;r)\colon t) \xrightarrow{a}_D \tilde{t}}{((s_1 [] s_2);r)\colon t \xrightarrow{a}_D \tilde{t}}. \tag{Backtrack}$$

The axiom (Elem) describes an elementary step. (Rec) embodies procedure execution by body replacement: for $x \Leftarrow g$ in D, execution of x amounts to execution of g. (Fail) replaces execution of $(\mathbf{fail};r)\colon t$ by that of t, its failure continuation. (Seq Comp) should be clear. (Backtrack) executes $((s_1 [] s_2);r)\colon t$ by executing $(s_1;r)$ and adding $(s_2;r)$ to the failure continuation t.

We shall now define how to obtain $\mathcal{O}$ from T_1. We need an auxiliary definition.

DEFINITION 3.5. Choose some fixed D.

a. Let $t_1, t_2 \in \mathcal{T}con$. The relation $t_1 \twoheadrightarrow t_2$ is the reflexive and transitive closure of the relation which holds between t_1 and t_2 whenever, for some $a \in A$ and $\tilde{t} \in \mathcal{T}con$, we have that

$$\frac{t_2 \xrightarrow{a}_D \tilde{t}}{t_1 \xrightarrow{a}_D \tilde{t}}$$

is a rule in T_1.

b. *t terminates* whenever $t \twoheadrightarrow E\colon t'$, for some t'

c. *t fails* whenever $t \twoheadrightarrow \Delta$.

The following lemma is immediate:

LEMMA 3.6. *For each t, either t terminates, or t fails, or, for some a, t', we have* $t \xrightarrow{a}_D t'$.

DEFINITION 3.7.

a. The mapping $\mathcal{O}\colon \mathcal{P}rog_1 \to R$ is given by

$$\mathcal{O}[\![<D|s>]\!] = \mathcal{O}_D[\![(s;E)\colon \Delta]\!]$$

b. The mapping Θ_D: $\mathscr{T}con_1 \to R$ is given by

$$\Theta[\![t]\!] = \epsilon, \text{ if } t \text{ terminates}$$
$$= \delta, \text{ if } t \text{ fails}$$
$$= a.\, \Theta_D[\![t']\!], \text{ if } t \xrightarrow{a}_D t'$$

where the transitions are with respect to T_1.

It may not be obvious that the function Θ_D is well- defined. This is in fact a consequence of the following

LEMMA 3.8. *Let the operator* Φ_D: $(\mathscr{T}con_1 \to R) \to (\mathscr{T}con_1 \to R)$ *be defined as follows: For any* $F \in \mathscr{T}con_1 \to R$, *we put*

$$\Phi_D(F)(t) = \epsilon, \quad \textit{if } t \textit{ terminates}$$
$$= \delta, \quad \textit{if } t \textit{ fails}$$
$$= a.F(t'), \quad \textit{if } t \xrightarrow{a}_D t'.$$

Then Φ_D *is a contracting mapping with* Θ_D *as its fixed point.*

PROOF. Clear from the definitions and Banach's theorem. □

The next step is the development of the denotational model. This model uses semantic counterparts for the syntactic continuations $\mathscr{R}con_1$ and $\mathscr{T}con_1$, in the form of

$(\phi \in) R \to R$, the (semantic) success continuations

$(v, w \in) R$, the (semantic) failure continuations

$(\pi \in) \mathbb{R} = (R \to R) \to R \to R$, a set which shall remain nameless.

Moreover, the usual notion of *environment* to deal with recursion is applied, this time in the form of $(\gamma \in) \Gamma_1$ defined as

$$\Gamma_1 = \mathbb{X} \to \mathbb{R}.$$

The denotational semantics function $\mathfrak{D}$ is of type

$$\mathfrak{D}: L_1 \to \Gamma_1 \to \mathbb{R}$$

i.e., it is well-defined to write $\mathfrak{D}[\![s]\!]\gamma\phi v = w$. The function $\mathfrak{D}$ will be used in the definition of $\mathfrak{M}$: $\mathscr{P}rog_1 \to R$.

From now on, we shall often suppress parentheses around arguments of functions. The denotational semantic definitions are collected in

DEFINITION 3.9 (denotational semantics for L_1, $\mathscr{P}rog_1$).

a. $\mathfrak{D}[\![a]\!]\gamma\phi v = a.\phi v$
$\mathfrak{D}[\![x]\!]\gamma\phi v = \gamma x \phi v$
$\mathfrak{D}[\![\textbf{fail}]\!]\gamma\phi v = v$
$\mathfrak{D}[\![s_1 ; s_2]\!]\gamma\phi v = \mathfrak{D}[\![s_1]\!]\gamma(\mathfrak{D}[\![s_2]\!]\gamma\phi)v$
$\mathfrak{D}[\![s_1 \,[]\, s_2]\!]\gamma\phi v = \mathfrak{D}[\![s_1]\!]\gamma\phi(\mathfrak{D}[\![s_2]\!]\gamma\phi v)$

b. $\mathfrak{M}$: $\mathscr{P}rog_1 \to R$ is given by $\mathfrak{M}[\![<D|s>]\!] = \mathfrak{D}[\![s]\!]\gamma_D(\lambda v.\epsilon)(\delta)$, with γ_D as in clause *c*

c. $\gamma_D = \gamma\{\pi_i / x_i\}_i$, where for $D \equiv <x_i \Leftarrow g_i>_i$.

$$<\pi_1, \cdots, \pi_n> = \text{fixed point } <\Phi_1, \cdots, \Phi_n>,$$

with Φ_j: $\mathbb{R}^n \to \mathbb{R}$ given by $\Phi_j(\pi'_1) \cdots (\pi'_n) = \mathfrak{D}[\![g_j]\!]\gamma\{\pi'_i / x_i\}_i$.

REMARKS.
1. In clause *a*, we assume as known the operation of prefixing a symbol a to an element w in R, yielding the result $a.w$.
2. Note the symmetry in the definitions of $\mathfrak{D}[\![s_1;s_2]\!]$ and $\mathfrak{D}[\![s_1 [] s_2]\!]$, where in the former case the success, in the latter case the failure continuation is extended.
3. In clause *b* the meaning of s is initialized with the empty success continuation $\lambda\nu.\epsilon$ and the empty failure continuation δ.
4. The (unique) fixed point in clause *c* exists by the guardedness requirement which ensures contractivity of the Φ_j (details of a proof of a very similar claim are provided in [BM]).

We continue with the derivation of the equivalence $\Theta=\mathfrak{M}$.

First, we introduce two auxiliary denotational meaning functions $\mathfrak{R}_D$ and $\mathfrak{T}_D$, working on elements in $\mathscr{R}con_1$ and $\mathscr{T}con_1$, respectively. (We find it convenient to carry along D as a parameter, rather than as explicit argument of the mappings $\mathfrak{R}$ and $\mathfrak{T}$.) The mappings $\mathfrak{R}_D$: $\mathscr{R}con_1 \to R \to R$ and $\mathfrak{T}_D$: $\mathscr{T}con_1 \to R$ are defined in

DEFINITION 3.10.
a. $\mathfrak{R}_D[\![E]\!]=\lambda\nu.\epsilon$, $\mathfrak{R}_D[\![s;r]\!]=\mathfrak{D}[\![s]\!]\gamma_D\mathfrak{R}_D[\![r]\!]$, with γ_D as in Definition 3.9
b. $\mathfrak{T}_D[\![\Delta]\!]=\delta$, $\mathfrak{T}_D[\![r:t]\!]=\mathfrak{R}_D[\![r]\!]\mathfrak{T}_D[\![t]\!]$.

The following lemma is now easily established (cf. Definition 3.5 for $\twoheadrightarrow$).

LEMMA 3.11. *Choose D fixed.*
a. $\mathfrak{T}_D[\![E:t]\!]=\epsilon$, $\mathfrak{T}_D[\![\Delta]\!]=\delta$
b. $\mathfrak{T}_D[\![(a;r):t]\!]=a.\mathfrak{T}_D[\![r:t]\!]$
c. If $t_1 \twoheadrightarrow t_2$, then $\mathfrak{T}_D[\![t_1]\!]=\mathfrak{T}_D[\![t_2]\!]$.

PROOF. We consider only one special case. Let $t_1 \equiv ((s_1 [] s_2);r)$: t, $t_2 \equiv (s_1;r)$: $((s_2;r):t)$. We have $\mathfrak{T}_D[\![t_1]\!]=\mathfrak{T}_D[\![((s_1 [] s_2);r)\colon t]\!]=\mathfrak{R}_D[\![(s_1 [] s_2);r]\!]\mathfrak{T}_D[\![t]\!]=\mathfrak{D}[\![s_1 [] s_2]\!]\gamma\mathfrak{R}_D[\![r]\!]\mathfrak{T}_D[\![t]\!]=$
$\mathfrak{D}[\![s_1]\!]\gamma_D\mathfrak{R}_D[\![r]\!](\mathfrak{D}[\![s_2]\!]\gamma_D\mathfrak{R}_D[\![r]\!]\mathfrak{T}_D[\![t]\!])= \cdots =\mathfrak{T}_D[\![(s_1;r)\colon ((s_2;r)\colon t)]\!]$. □

The key step in the proof that $\Theta=\mathfrak{M}$ holds on $\mathscr{P}rog_1$ is the following lemma (which constitutes an application to L_1 of the general proof techniques of [KR], [BM]):

LEMMA 3.12. *Let* Φ_D: $(\mathscr{T}con_1 \to R) \to (\mathscr{T}con_1 \to R)$ *be defined as follows (cf. Lemma 3.8). Let* $F \in \mathscr{T}con_1 \to R$. *We put*

$$\begin{aligned}\Phi_D(F)(t) &= \epsilon, \ \textit{if } t \textit{ terminates}\\ &= \delta, \ \textit{if } t \textit{ fails}\\ &= a.F(t'), \ \textit{if } t \xrightarrow{a}_D t'\end{aligned}$$

The $\Phi_D(\mathfrak{T}_D)=\mathfrak{T}_D$.

PROOF. We introduce the following *complexity measure* c on $t \in \mathscr{T}con_1$: $c(E:t)=c(\Delta)=0$, $c((s;r)\colon t)=c(s)+c(t)$, $c(a)=c(x)=c(\textbf{fail})=1$, $c(s_1;s_2)=c(s_1 [] s_2)=c(s_1)+c(s_2)+1$. From the definition of T_1 we see that, for each t_1, t_2 such that
(i) $t_1 \twoheadrightarrow t_2$,
(ii) $t_1 \neq t_2$, and
(iii) t_1 of the form $(g;r)\colon t'$
we have that $c(t_1)>c(t_2)$.

We now prove that $\Phi_D(\mathfrak{T}_D)(t)=\mathfrak{T}_D[\![t]\!]$, for each t. If t terminates or t fails, the result is clear by definition. Otherwise, t is of the form $t \equiv (s;r)\colon t'$, and, for some a, t_0, we have $t \xrightarrow{a}_D t_0$. We organize

the proof in two stages. Let us call a failure continuation t guarded if $t=\Delta$ or if t is of the form $(g;r):t'$, with t' guarded. We first consider the case of a guarded t of the form $(g;r):t'$, and prove the result by induction on $c((g;r):t')$. The following cases are distinguished:

$g \equiv a$. $\Phi_D(\mathfrak{T}_D)((a;r): t') = (\text{def. } \Phi_D) a.\mathfrak{T}_D[\![r:t']\!] = (\text{Lemma 3.11})\ \mathfrak{T}_D[\![(a;r): t']\!]$.

$g \equiv \textbf{fail}$. $\Phi(\mathfrak{T}_D)((\textbf{fail};r): t') = (\text{def. } \Phi_D)\Phi_D(\mathfrak{T}_D)(t') = (\text{ind. hyp.})\ \mathfrak{T}_D[\![t']\!] = (\text{Lemma 3.11})\ \mathfrak{T}_D[\![(\textbf{fail};r): t']\!]$.

$g \equiv (g_1;s)$.

$$\begin{aligned}
\Phi_D(\mathfrak{T}_D)(((g_1;s);r): t') &= (\text{def. } \Phi_D)\\
\Phi_D(\mathfrak{T}_D)((g_1;(s;r)): t') &= (\text{ind. hyp.})\\
\mathfrak{T}_D[\![(g_1;(s;r)): t']\!] &= (\text{Lemma 3.11})\\
\mathfrak{T}_D[\![((g_1;s);r): t']\!]
\end{aligned}$$

$g \equiv (g_1 [] g_2)$. Similar

As next stage, we prove the desired result for t of the form $(s;r): t'$, for any $s \in L_1$ and any t'. The structure of the argument is as in stage 1, but for the case $s \equiv x$. We then have

$$\begin{aligned}
\Phi_D(\mathfrak{T}_D)((x;r): t') &= (\text{def. } \Phi_D)\\
\Phi_D(\mathfrak{T}_D)((g;r): t') &= (\text{stage 1})\\
\mathfrak{T}_D[\![(g;r): t']\!] &= (\text{Lemma 3.11})\\
\mathfrak{T}_D[\![(x;r): t']\!] & \qquad \square
\end{aligned}$$

The main result of the present section is now a direct consequence of this lemma:

THEOREM 3.13. *For each* $\sigma \in \mathscr{Prog}_1$, $\mathcal{O}[\![\sigma]\!] = \mathcal{N}[\![\sigma]\!]$.

PROOF. By Lemma 3.8 and Lemma 3.12, Φ_D is a contracting mapping, hence its fixed points $\mathcal{O}_D$ and $\mathfrak{T}_D$ coincide. Now

$$\mathcal{O}[\![\sigma]\!] = \mathcal{O}[\![<D|s>]\!] = \mathcal{O}_D[\![(s;E): \Delta]\!] = \mathfrak{T}_D[\![(s;E): \Delta]\!] = \mathcal{N}[\![s]\!]\gamma_D(\lambda\nu.\epsilon)(\delta) =$$
$$\mathcal{N}[\![<D|s>]\!] = \mathcal{N}[\![\sigma]\!]. \quad \square$$

We conclude this section with the discussion of an alternative definition for the denotational semantics for L_1. We do this to prepare the way for an understanding of a definition using similar techniques in Section 5. Whereas in the present context the new definition is no more than a (maybe somewhat far—fetched) alternative, in Section 5 the situation will be such that a definition following the pattern as presented in a minute will be the only one possible.

We recall our use of the set $\mathbb{R} = (R \to R) \to R \to R$. It is now necessary to be more precise about the various functions encountered in $\mathbb{R}$. Therefore, till the end of this section we take $\mathbb{R}$ as

$$\mathbb{R} = (R \to^1 R) \to^1 R \to^1 R$$

(See Definition 2.3.4 for the $\to^1$ notation for functions.)

DEFINITION 3.14. Let Ψ_D be a mapping

$$\Psi_D: (L_1 \to \mathbb{R}) \to (L_1 \to \mathbb{R})$$

which we define by putting, for each fixed D, each $F \in L_1 \to \mathbb{R}$, each $\phi \in R \to^1 R$, and each $\nu \in R$:

$$\Psi_D F a \phi \nu = a.\phi\nu$$

$$\Psi_D F x \phi \nu = \Psi_D F g \phi \nu, \quad \text{for } x \Leftarrow g \text{ in } D$$

$$\Psi_D F \textbf{fail} \phi \nu = \nu$$

$$\Psi_D F(s_1;s_2)\phi\nu = \Psi_D Fs_1(Fs_2\phi)\nu$$

$$\Psi_D F(s_1 [] s_2)\phi\nu = \Psi_D Fs_1\phi(\Psi_D Fs_2\phi\nu)$$

Moreover, we put $\mathcal{D}_D =$ fixed point Ψ_D.

We show that Ψ_D is well-defined and contracting in F, hence justifying the definition of $\mathcal{D}_D$. This follows by the following

THEOREM 3.15. *Choose D fixed (and then suppress it in the notation). Let* $F, F_1, F_2 \in L_1 \to \mathbb{R}$, ϕ, ϕ_1, $\phi_2 \in R \to^1 R$, $\nu, \nu_1, \nu_2 \in R$.

a1. *For all* $g \in L_1^g$.

$$d(\Psi Fg\phi\nu_1, \Psi Fg\phi\nu_2) \leqslant d(\nu_1, \nu_2)$$

a2. *For all* $s \in L_1$

$$d(\Psi Fs\phi\nu_1, \Psi Fs\phi\nu_2) \leqslant d(\nu_1, \nu_2)$$

b1. *For all* $g \in L_1^g$

$$d(\Psi Fg\phi_1, \Psi Fg\phi_2) \leqslant \tfrac{1}{2} d(\phi_1, \phi_2)$$

b2. *Similarly for* $s \in L_1$

c1. *For all* $g \in L_1^g$

$$d(\Psi F_1 g, \Psi F_2 g) \leqslant \tfrac{1}{2} d(F_1, F_2)$$

c2. *Similarly for* $s \in L_1$.

PROOF. We exhibit various selected subcases.

a1. $g \equiv$ **fail**. $d(\Psi F\mathbf{fail}\phi\nu_1, \Psi F\mathbf{fail}\phi\nu_2) = d(\nu_1, \nu_2)$

$g \equiv g_1;s$. $d(\Psi F(g_1;s)\phi\nu_1, \Psi F(g_1;s)\phi\nu_2) = d(\Psi Fg_1(Fs\phi)\nu_1, \Psi Fg_1(Fs\phi)\nu_2)$

$\leqslant$(ind. hyp., and since $Fs\phi \in R \to^1 R$) $d(\nu_1, \nu_2)$

a2. $s \equiv x$. $d(\Psi Fx\phi\nu_1, \Psi Fx\phi\nu_2) = d(\Psi Fg\phi\nu_1, \Psi Fg\phi\nu_2)$ (with $x \Leftarrow g$ in D) $\leqslant d(\nu_1, \nu_2)$ by part a1.

b1. $g \equiv a$. $d(\Psi Fa\phi_1, \Psi Fa\phi_2) = \sup_{\nu \in R} d(\Psi Fa\phi_1\nu, \Psi Fa\phi_2\nu) = \sup_{\nu \in R} d(a.\phi_1\nu, a.\phi_2\nu) =$

$\tfrac{1}{2}\sup_{\nu \in R} d(\phi_1\nu, \phi_2\nu) = \tfrac{1}{2} d(\phi_1, \phi_2)$

$g \equiv g_1 [] g_2$. $d(\Psi F(g_1 [] g_2)\phi_1, \Psi F(g_1 [] g_2)\phi_2) = \sup_{\nu \in R} d(\Psi F(g_1 [] g_2)\phi_1\nu, \Psi F(g_1 [] g_2)\phi_2\nu) =$

$\sup_{\nu \in R} d(\Psi Fg_1\phi_1(\Psi Fg_2\phi_1\nu), \Psi Fg_1\phi_2(\Psi Fg_2\phi_2\nu)) \leqslant$(d is an ultrametric)

$\sup_{\nu \in R} \max(d(\Psi Fg_1\phi_1(\Psi Fg_2\phi_1\nu), \Psi Fg_1\phi_1(\Psi Fg_2\phi_2\nu)),$

$d(\Psi Fg_1\phi_1(\Psi Fg_2\phi_2\nu), \Psi Fg_1\phi_2(\Psi Fg_2\phi_2\nu)))$

$\leqslant$ (part a1, ind. on g_1)

$\sup_{\nu \in R} \max(d(\Psi Fg_2\phi_1\nu, \Psi Fg_2\phi_2\nu), \tfrac{1}{2} d(\phi_1, \phi_2))$

$\leqslant$(ind. on g_2) $\tfrac{1}{2} d(\phi_1, \phi_2)$.

b2. Similar to a2.

c1. $g \equiv g_1;s$. $d(\Psi F_1(g_1;s), \Psi F_2(g_1;s)) = \sup_{\phi \in R \to^1 R, \nu \in R} d(\Psi F_1(g_1;s)\phi\nu, \Psi F_2(g_1;s)\phi\nu)$

$\leqslant$(d is an ultrametric)

$\sup_{\phi\in R\to{}^1R,v\in R}\max(d(\Psi F_1g_1(F_1s\phi)v,\Psi F_1g_1(F_2s\phi)v),d(\Psi F_1g_1(F_2s\phi)v,\Psi F_2g_1(F_2s\phi)v))$

$\leqslant$ (part b, ind. hyp. for g_1)

$\sup_{\phi\in R\to{}^1R,v\in R}\max(\frac{1}{2}d(F_1s\phi,F_2s\phi),\frac{1}{2}d(F_1,F_2))\leqslant\frac{1}{2}d(F_1,F_2)$

$g\equiv g_1[]g_2$. $d(\Psi F_1(g_1[]g_2),\Psi F_2(g_1[]g_2))=\sup_{\phi\in R\to{}^1R,v\in R}d(\Psi F_1(g_1[]g_2)\phi v,\Psi F_2(g_1[]g_2)\phi v)$

$=\sup_{\phi\in R\to{}^1R,v\in R}d(\Psi F_1g_1\phi(\Psi F_1g_2\phi v),\Psi F_2g_1\phi(\Psi F_2g_2\phi v))\leqslant$

(similar to part b1, using the ultrametric property, twice the ind. hyp., and part a)

$\frac{1}{2}d(F_1,F_2)$. □

COROLLARY 3.16. *For each* $s\in L_1$, $\mathfrak{D}_D[\![s]\!]=\mathfrak{D}[\![s]\!]\gamma_D$.

4. SEQUENTIAL LOGIC PROGRAMMING WITH BACKTRACKING AND ABSOLUTE CUT

We add a preliminary version of the cut operator, written as '!' and inspired by PROLOG's cut, to the language L_1, obtaining L_2. This version we call 'absolute cut'. Its operation is rather drastic: when the operator '!' is encountered, all alternatives (kept available for possible subsequent backtracking through a fail statement) collected as a result of previous executions of $s_1[]s_2$-statements *since the beginning of the whole program are deleted.* In the next section we shall deal with a more realistic version of the cut operator, denoted by '!!' and called 'relative cut'. The operator '!!' deletes all alternatives (kept available for possible subsequent backtracking through a fail statement) collected as a result of previous executions of $s_1[]s_2$-statements *since the beginning of the execution of the most recent procedure call in which this '!!' occurs.* We emphasize that the !!-operator is the one which interests us. The '!' is studied only to help in understanding our treatment of '!!' in the next section. In particular, the mechanism developed in the present section introducing the so-called dump stack is not so much motivated by our wish to model '!' (in fact, all applications of transition system T_2 leave the dump stack constant), but rather designed for modelling '!!' (in T_3 the dump stack indeed varies).

We shall design operational and denotational models for L_2 (and for L_3 in the next section) involving a more subtle use of continuations. The operational semantics for L_3 (and its approximation L_2) are due to De Vink, cf. [Vi, BrVi]. Our continuation based denotational semantics for L_2 will be designed such that the equivalence $\mathfrak{O}=\mathfrak{D}$ on $\mathscr{Prog}_2$ is a straightforward extension of the results in Section 3.

DEFINITION 4.1. (syntax)

a. (statements). The class ($s\in$)L_2 of *statements* is given by

$$s::= a|x|\textbf{fail}|s_1;s_2|s_1[]s_2|!$$

b,c,d. The classes L_2^g, $\mathscr{Decl}_2$, $\mathscr{Prog}_2$ are derived from L_2 analogously to Definition 3.1, parts b,c,d. We now present the new continuations:

DEFINITION 4.2.

a. The class ($u\in$)$\mathscr{Ucon}$ of *statement continuations* is defined by

$$u::= \textbf{nil}|(s;u)$$

b. The class ($r\in$)$\mathscr{Rcon}_2$ of *success continuations* is defined by

$$r::= E|(u:t);r$$

c. The class $(t \in)\mathscr{T}con_2$ of *failure continuations* is defined by

$$t :: = \Delta \mid (r : t)$$

As before we define a transition system is terms of fourtuples in $\mathscr{T}con_2 \times A \times \mathscr{D}ecl_2 \times \mathscr{T}con_2$, employing the notation

$$t \overset{a}{\rightarrow}_D t'$$

DEFINITION 4.3 (transition system T_2).

$$(((a;u): t);r): t' \overset{a}{\rightarrow}_D ((u:t);r): t' \qquad \text{(Elem)}$$

$$\frac{r : t' \overset{a}{\rightarrow}_D \tilde{t}}{((\mathbf{nil}:t);r): t' \overset{a}{\rightarrow}_D \tilde{t}} \qquad \text{(Nil)}$$

$$\frac{(((g;u): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}}{(((x;u): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}}, \quad x \Leftarrow g \text{ in } D \qquad \text{(Rec)}$$

$$\frac{t' \overset{a}{\rightarrow}_D \tilde{t}}{(((\mathbf{fail};u): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}} \qquad \text{(Fail)}$$

$$\frac{(((s_1;(s_2;u)): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}}{((((s_1;s_2);u): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}} \qquad \text{(Seq Comp)}$$

$$\frac{(((s_1;u): t);r): ((((s_2;u): t);r): t') \overset{a}{\rightarrow}_D \tilde{t}}{((((s_1 \,[]\, s_2);u): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}} \qquad \text{(Backtrack)}$$

$$\frac{((u:t);r): t \overset{a}{\rightarrow}_D \tilde{t}}{(((!;u): t);r): t' \overset{a}{\rightarrow}_D \tilde{t}} \qquad \text{(Cut)}$$

It may be instructive to compare T_2 with T_1. The system is organized by the various cases for s in $t_0 \equiv (((s;u): t);r): t'$. In t_0, t' is the failure continuation, also to be called failure *stack*, which serves the same purpose as in the constructs $(s;r): t'$ encountered in T_1. On the other hand, t in t_0 is the 'dump stack' (terminology from [Vi]). Its function is as follows: When, as a result of (Backtrack) some $t_0 \equiv ((((s_1 \,[]\, s_2);u): t);r): t'$ is transformed (by $\gg$) into $t_1 \equiv (((s_1;u): t);r): \bar{t}$ (where $\bar{t} \equiv (((s_2;u): t);r): t'$), in t_1 the stack t is preserved. In case we encounter, while processing $s_1;u$, an occurrence of '!', we shall transform the then current $t'' \equiv (((!;u'): t);r): \bar{\bar{t}}$ into $((u':t);r): t$, thus reinstalling the dump stack t instead of the currently active failure stack $t_=$, effectively throwing away the alternatives built up so far in $t_=$ as a result of the []-statements processed up to now. The other rules in T_2 should be clear: Once the formalism involving a second (dump) stack is understood, the axioms (Elem) and the rules (Rec), (Fail), (Seq Comp) and (Backtrack) are direct extensions of similar rules in T_1. Rule (Nil) expresses the natural fact that execution of $((\mathbf{nil}: t);r): t'$ amounts to execution of $r: t'$. We already announce that in transition system T_3 dealing with relative cut, we shall *only* vary the recursion rule (and replace '!' by '!!' in the (Cut) rule).

We next define how to obtain Θ from T_2. The notions of terminating or failing t are as in

Definition 3.5.

DEFINITION 4.4.

a. The mapping Θ: $\mathscr{Prog}_2 \to R$ is given by

$$\Theta[\![<D|s>]\!] = \Theta_D[\![(((s;\mathbf{nil}):\Delta);E):\Delta]\!]$$

b. The mapping Θ_D: $\mathscr{Tcon}_2 \to R$ is given by

$$\Theta_D[\![t]\!] = \epsilon, \text{ if } t \text{ terminates}$$
$$= \delta, \text{ if } t \text{ fails}$$
$$= a.\Theta_D[\![t']\!], \text{ if } t \xrightarrow{a}_D t'$$

where the transitions are with respect to T_2.

Well- definedness of Θ for $\mathscr{Prog}_2$ follows as before.

We continue with the denotational semantics. Following the general strategy as first adopted in the previous section, we shall structure the denotational definitions in direct correspondence with the operational ones in T_2. We first introduce the various domains

$$(v, w \in)R$$
$$(\phi \in)R \to R$$
$$(\rho \in)R \to (R \to R) \to R \to R$$
$$(\pi \in)(R \to (R \to R) \to R \to R) \to (R \to (R \to R) \to R \to R) \stackrel{af}{=} \mathbb{R}$$
$$(\gamma \in)\Gamma_2 = \mathbb{X} \to \mathbb{R}$$

The mappings $\mathfrak{D}$: $L_2 \to \Gamma_2 \to \mathbb{R}$ and $\mathfrak{M}$: $\mathscr{Prog}_2 \to R$ are given in

DEFINITION 4.5.

a. $\mathfrak{D}[\![a]\!]\gamma\rho\nu\phi w = a.\rho\nu\phi w$
$\mathfrak{D}[\![x]\!]\gamma\rho\nu\phi w = \gamma x \rho\nu\phi w$
$\mathfrak{D}[\![\mathbf{fail}]\!]\gamma\rho\nu\phi w = w$
$\mathfrak{D}[\![s_1;s_2]\!]\gamma\rho\nu\phi w = \mathfrak{D}[\![s_1]\!]\gamma(\mathfrak{D}[\![s_2]\!]\gamma\rho)\nu\phi w$
$\mathfrak{D}[\![s_1 [] s_2]\!]\gamma\rho\nu\phi w = \mathfrak{D}[\![s_1]\!]\gamma\rho\nu\phi(\mathfrak{D}[\![s_2]\!]\gamma\rho\nu\phi w)$
$\mathfrak{D}[\![!]\!]\gamma\rho\nu\phi w = \rho\nu\phi\nu$

b. $\gamma_D = \gamma\{\pi_i / x_i\}_i$, where $<\pi_1, \cdots, \pi_n>$ is the (unique) fixed point derived in the usual way from $D \equiv <x_i \Leftarrow g_i>_i$

c. $\mathfrak{M}[\![<D|s>]\!] = \mathfrak{D}[\![s]\!]\gamma_D(\lambda\nu.\lambda\phi.\phi)(\delta)(\lambda v.\epsilon)(\delta)$

REMARK. The definitions in part a follow the axiom and rules in T_2. The following correspondence is maintained:

$$u \in \mathscr{Ucon} \Leftrightarrow \rho \in R \to (R \to R) \to R \to R$$
$$r \in \mathscr{Rcon}_2 \Leftrightarrow \phi \in R \to R$$
$$t, t' \in \mathscr{Tcon}_2 \Leftrightarrow v, w \in R$$

Also, a construct $((u:t);r)$: t' corresponds with the semantic entity $\rho\nu\phi w$.

Similar to what we did in Section 3, on our way to establish that $\Theta = \mathfrak{M}$ we use some (auxiliary) denotational semantic functions $\mathfrak{U}_D$, $\mathfrak{R}_D$, $\mathfrak{T}_D$ with types

$$\mathfrak{U}_D : \mathscr{Ucon} \to R \to (R \to R) \to R \to R$$

$$\mathcal{R}_D : \mathcal{T}con_2 \to R \to R$$

$$\mathcal{T}_D : \mathcal{T}con_2 \to R$$

defined in

DEFINITION 4.6.
a. $\mathcal{U}_D[\![\mathbf{nil}]\!] = \lambda\nu.\lambda\phi.\phi$
$\mathcal{U}_D[\![s;u]\!] = \mathcal{N}[\![s]\!]\gamma_D\mathcal{U}_D[\![u]\!]$
b. $\mathcal{R}_D[\![E]\!] = \lambda\nu.\epsilon$
$\mathcal{R}_D[\![(u:t);r]\!] = \mathcal{U}_D[\![u]\!]\mathcal{T}_D[\![t]\!]\mathcal{R}_D[\![r]\!]$
c. $\mathcal{T}_D[\![\Delta]\!] = \delta$
$\mathcal{T}_D[\![r:t]\!] = \mathcal{R}_D[\![r]\!]\mathcal{T}_D[\![t]\!]$

The following lemma is now easily established:

LEMMA 4.7. *Choose D fixed.*
a. $\mathcal{T}_D[\![E: t]\!] = \epsilon$, $\mathcal{T}_D[\![\Delta]\!] = \delta$
b. *If* $t_1 \twoheadrightarrow t_2$, *then* $\mathcal{T}_D[\![t_1]\!] = \mathcal{T}_D[\![t_2]\!]$

PROOF. Clear from the definitions. □

We finally have

THEOREM 4.8. *Let* Φ_D: $(\mathcal{T}con_2 \to R) \to (\mathcal{T}con_2 \to R)$ *be defined as follows. Let* $F \in \mathcal{T}con_2 \to R$.

$$\begin{aligned}\Phi_D(F)(t) &= \epsilon, \text{ if } t \text{ terminates}\\ &= \delta, \text{ if } t \text{ fails}\\ &= a.F(t'), \text{ if } t \xrightarrow{a}_D t'\end{aligned}$$

Then $\Phi_D(\mathcal{T}_D) = \mathcal{T}_D$.

PROOF. We employ the following complexity measure

$$c(E: t) = c(\Delta) = c(\mathbf{nil}) = 0$$

$$c(((\mathbf{nil}: t);r): t') = 1 + c(r: t')$$

$$c(((u:t);r): t') = c(u) + \max(c(t), c(t')), \quad u \neq \mathbf{nil}$$

$$c(s;u) = c(s) + c(u),\ c(s_1;s_2) = c(s_1 \,[]\, s_2) = 1 + c(s_1) + c(s_2),$$

$$c(a) = c(x) = c(\mathbf{fail}) = c(!) = 1$$

Let us call t guarded if either $t = \Delta$, or t is of the form $(((g;u): t');r): t''$ or $((\mathbf{nil}: t');r): t''$, with t' and t'' guarded. From the definition of T_2 we can infer that, for guarded t_1, if $t_1 \twoheadrightarrow t_2$ then we have that $c(t_1) > c(t_2)$. Now follow the same argument as in the proof of Lemma 3.12. □

COROLLARY 4.9. *For each* $\sigma \in \mathcal{P}rog_2$, $\mathcal{O}[\![\sigma]\!] = \mathcal{M}[\![\sigma]\!]$.

PROOF. Cf. the proof of Theorem 3.13. □

5. Sequential logic programming with backtracking and relative cut

Thanks to the preparations in Sections 3 and 4, we can now be quite brief. In L_3, we replace '!' by '!!', and assume all induced syntactic definitions. We define the transition system T_3 in

Definition 5.1. T_3 coincides with T_2 (with !! replacing ! in the (Cut) rule), but for the rule (Rec) of T_2 which is now replaced by

$$\frac{(((g\,;\mathbf{nil})\colon t');((u\colon t);r))\colon t' \overset{a}{\to}_D \tilde{t}}{(((x\,;u)\colon t);r)\colon t' \overset{a}{\to}_D \tilde{t}}, \quad x \Leftarrow g \text{ in } D \qquad \text{(Rec')}$$

As a result of (Rec'), if $t_0 \equiv (((x\,;u)\colon t);r)\colon t' \gg t_1 \equiv (((g\,;\mathbf{nil})\colon t');((u\colon t);r))\colon t'$, with $x \Leftarrow g$ in D, u keeps its dump stack t, but the dump stack for g is initialized at the current failure stack t'. As a consequence, occurrences of !! in g cause (re) activation of t' as failure stack rather than that of t.

From T_3 the operational semantics definitions for L_3 and $\mathcal{P}rog_3$ are obtained in the, by now usual, way.

We now discuss how to design the denotational semantics for L_3. We want to follow the general strategy (denotational equations derived from the transition system), but then face a complication in the new rule for procedures (Rec'): A call of some x does *not* simply amount to body replacement (g replacing x in the current t), since, in addition, various parameters are changed around. Thus, a direct definition involving $\gamma_D = \gamma\{\pi_i/x_i\}_i$ as in the previous section, with $<\pi_i>_i$ fixed points derived from the declaration $D \equiv <x_i \Leftarrow g_i>_i$ is not feasible. Rather, we follow the route as described at the end of Section 3, and obtain $\mathfrak{D}$ as fixed point of a higher order operator Ψ_D. In the equations defining this operator, we once more can mimick the axioms and rules of transition system T_3. As we did at the end of Section 3, we add to our various domains the information that all relevant functions are (at least) non distance increasing:

$$(v, w \in)R$$

$$(\phi \in)R \to^1 R$$

$$(\rho \in)R \to^1 (R \to^1 R) \to^1 R \to^1 R$$

$$(\pi \in)(R \to^1 (R \to^1 R) \to^1 R \to^1 R) \to^1 (R \to^1 (R \to^1 R) \to^1 R \to^1 R) = \mathbb{R}$$

We define $\Psi_D\colon (L_3 \to \mathbb{R}) \to (L_3 \to \mathbb{R})$ in

Definition 5.2.

a. Let $F \in L_3 \to \mathbb{R}$. (In part a, we suppress subscripts D)

$$\Psi F a \rho v \phi w = a.\rho v \phi w$$

$$\Psi F x \rho v \phi w = \Psi F g (\lambda v.\lambda \phi.\phi) w (\rho v \phi) w, \quad x \Leftarrow g \text{ in } D$$

$$\Psi F \,\mathbf{fail}\, \rho v \phi w = w$$

$$\Psi F (s_1\,;s_2) \rho v \phi w = \Psi F s_1 (F s_2 \rho) v \phi w$$

$$\Psi F (s_1 \,[]\, s_2) \rho v \phi w = \Psi F s_1 \rho v \phi (\Psi F s_2 \rho v \phi w)$$

$$\Psi F (!!) \rho v \phi w = \rho v \phi v$$

b. $\mathfrak{D}_D =$ fixed point (Ψ_D), $\mathfrak{M}[\![<D|s>]\!] = \mathfrak{M}_D[\![s]\!]$, $\mathfrak{M}_D = \lambda s.\mathfrak{D}_D[\![s]\!](\lambda v.\lambda \psi.\psi)(\delta)(\lambda v.\epsilon)(\delta)$.

We show that Ψ_D is well-defined and contracting in D, hence justifying the definition of $\mathfrak{D}_D$.

Theorem 5.3.

a. $d(\Psi F g \rho v \phi w_1, \Psi F g \rho v \phi w_2) \leqslant d(w_1, w_2)$, *for all* $g \in L_3^g$, *and similarly with* $s(\in L_3)$ *replacing* g

b. $d(\Psi F g \rho v \phi_1, \Psi F g \rho v \phi_2) \leqslant \frac{1}{2} d(\phi_1, \phi_2)$, *for all* $g \in L_3^g$, *and similarly with* $s(\in L_3)$ *replacing* g

c. $d(\Psi Fg\rho v_1, \Psi Fg\rho v_2) \leqslant d(v_1, v_2)$, *for all* $g \in L_3^g$, *and similarly with* $s(\in L_3)$ *replacing* g

d. $d(\Psi Fg\rho_1, \Psi Fg\rho_2) \leqslant \frac{1}{2} d(\rho_1, \rho_2)$, *for all* $g \in L_3^g$, *and similarly with* $s(\in L_3)$ *replacing* g

e. $d(\Psi F_1 g, \Psi F_2 g) \leqslant \frac{1}{2} d(F_1, F_2)$, *for all* $g \in L_3^g$, *and similarly with* $s(\in L_3)$ *replacing* g.

PROOF. The proof is very similar to that of Theorem 3.15. We consider just two subcases. The first concerns clause d, subcase $g \equiv g_1; s$. We have

$$d(\Psi F(g_1;s)\rho_1, \Psi F(g_1;s)\rho_2) = d(\Psi Fg_1(Fs\rho_1), \Psi Fg_1(Fs\rho_2)) \leqslant \text{(ind. hyp.)}$$
$$\frac{1}{2} d(Fs\rho_1, Fs\rho_2) \leqslant \text{(def. } F) \frac{1}{2} d(\rho_1, \rho_2)$$

Next, we consider clause d, case $s \equiv x$. We have

$$d(\Psi Fx\rho_1, \Psi Fx\rho_2) = \sup_{v,w \in R,\, \phi \in R \to^1 R} d(\Psi Fg(\lambda v.\lambda\psi.\psi)w(\rho_1 v\phi), \Psi Fg(\lambda v.\lambda\psi.\psi)w(\rho_2 v\phi)) \leqslant \text{(part b)}$$
$$\sup_{v \in R,\, \phi \in R \to^1 R} \frac{1}{2} d(\rho_1 v\phi, \rho_2 v\phi) \leqslant \frac{1}{2} d(\rho_1, \rho_2) \quad \square$$

Now that we have justified the definition of $\mathfrak{D}_D$, by the usual argument we immediately obtain

COROLLARY 5.4. *For each* $\sigma \in \mathscr{Prog}_3$, $\mathfrak{O}[\![\sigma]\!] = \mathfrak{M}[\![\sigma]\!]$.

6. (AND/OR) PARALLEL LOGIC PROGRAMMING: THE LINEAR TIME CASE

We next turn our attention to the imperative features underlying the general model of logic programming (rather than the PROLOG-like variant discussed so far). Accordingly, we now allow parallel execution, and, moreover, replace the backtracking choice $s_1 [] s_2$ (don't know) by the general nondeterministic choice $s_1 + s_2$ (don't care). We shall find it advantageous to also keep sequential composition in our language. Parallel execution will be taken here in the interleaving sense: The favorite example is $a \| b$, which obtains as meaning the set $\{ab, ba\}$. Thus, we have a computational model which allows, in general, many outcomes of a computation, and sets rather than single elements are yielded as a result of the semantic mappings.

The simultaneous presence of sequences of elementary actions and of (an element modelling) failure in the sets of entities which are the meaning of a statement or program leads to the following well-known phenomenon (we use $v, w \in R$ as before but now also consider subsets $X, Y \subseteq R$): Firstly, if there is a choice between failure or something else (some $X \subseteq R$), we want to keep only the something else:

$$\{\delta\} \cup X = X, \quad \text{for} \quad X \neq \varnothing \tag{6.1}$$

Secondly, we want that, for any v,

$$\delta.v = \delta \tag{6.2}$$

(no visible result after failure), but we do not want that $v.\delta = \delta$, for all v. That is, we do not want that failure collapses all previous results. The last property explains that it is not adequate to simply model failure by the empty subset of R, since we do have the $v.\varnothing = \varnothing$, for all $v \in R$. Note that this argument depends on the interpretation of '.' as the usual concatenation operator; in a moment, we shall discuss an alternative interpretation for '.'. Thirdly, we have the choice (in our semantic model) as to how to model the interplay between failure and sequence formation. In the present section we shall take the sequencing operator in the usual sense of concatenation ('.') of sequences of symbols, and treat δ as a special symbol satisfying (6.1) and (6.2). Accordingly, we then have that

$$v.X_1 \cup v.X_2 = v.(X_1 \cup X_2) \tag{6.3}$$

with as corollary that $v.\{\delta\} \cup v.X = v.X$, for $X \neq \varnothing$. By the semantic definitions to follow, (6.3) is at the bottom of the equivalence

$$(s;s_1)+(s;s_2) = s;(s_1+s_2) \tag{6.4}$$

In a variety of phrasings stemming from different sources, we say something like
- sequential composition is left-distributive with respect to nondeterministic choice
- we have a 'linear time' or trace model for the denotational semantics
- the nondeterminacy is local or internal.

In the next section, we shall adopt an alternative view, and use a different operator for sequence formation, denoted by ':', which does *not* satisfy $(v:X_1)\cup(v:X_2)=v:(X_1\cup X_2)$. We then obtain a model in which it is not, in general, true that $s:(s_1+s_2)$ and $(s:s_1)+(s:s_2)$ have the same meaning. This model, to be described in detail in Section 7, is called 'branching time', and the operator ':' is, in the framework of parallel logic programming languages, called (don't care) *commit*. In Section 8, finally, we shall investigate what happens when we combine the two sequential operators ';' and ':' into one language. This will give rise to some interesting ensuing problems which can, somewhat metaphorically, be described as having to do with the *grain size* of atoms in computations. Most of the material in Sections 6,7 is essentially known. The semantic definitions go back to papers such as [BMOZ, BKMOZ], and the equivalence proofs are versions of the results in [KR] (the syntactic format for recursion adopted in [KR] causes some technical complications not encountered below) or [BM]. What may be new is the emphasis on the comparative analysis of ';' versus ':' (without involving different versions of nondeterminacy). The idea to investigate properties of the commit operator as a semantic operator in a branching time framework is due to Kok [Ko].

After these explanations, we can be rather concise in the subsequent definitions.

Definition 6.1. (Syntax for L_4).

a (statements). The class of statements $(s\in)L_4$ is given by

$$s::= a|x|\mathbf{fail}|s_1;s_2|s_1\|s_2|s_1+s_2$$

b (guarded statements). The class of guarded statements $(g\in)L_4^g$ is given by

$$g::= a|\mathbf{fail}|g;s|g_1\|g_2|g_1+g_2$$

c. $(D\in)\mathscr{D}ecl_4$ and $(\sigma\in)\mathscr{P}rog_4$ are as usual.

From now on, we take $R=A^+\cup A^\omega\cup A^*.\{\delta\}$: We have no more use for $\epsilon\in R$.

Definition 6.2. $\mathbb{S}=\mathcal{P}_{nc}(R)$ is the set of all nonempty closed subsets of R.

The metric framework employed below relies on

Lemma 6.3. *Let $\hat{d}$ be the Hausdorff distance on $\mathbb{S}$. Then $(\mathbb{S},\hat{d})$ is a complete ultrametric space.*

Proof. See, e.g., [Ni]. □

Below, we shall assume as known the operation of prefixing $a\in A$ to $X\in\mathbb{S}$ yielding $a.X\in\mathbb{S}$.

The transition system T_4 is defined in terms of transitions in $L_4\times A\times\mathscr{D}ecl_4\times(L_4\cup\{E\})$, written as

$$s\xrightarrow{a}_D s'$$

or

$$s\xrightarrow{a}_D E.$$

Definition 6.4 (transition system T_4). Let t range over $L_4\cup\{E\}$.

$$a \xrightarrow{a}_D E \qquad \text{(Elem)}$$

$$\frac{g \xrightarrow{a}_D t}{x \xrightarrow{a}_D t}, \quad x \Leftarrow g \quad \text{in} \quad D \qquad \text{(Rec)}$$

$$\frac{s \xrightarrow{a}_D s' | E}{s;\bar{s} \xrightarrow{a}_D s';\bar{s} | \bar{s}} \qquad \text{(Seq Comp)}$$

$$s \| \bar{s} \xrightarrow{a}_D s' \| \bar{s} | \bar{s} \qquad \text{(Par Comp)}$$

$$\bar{s} \| s \xrightarrow{a}_D \bar{s} \| s' | \bar{s}$$

$$\frac{s \xrightarrow{a}_D t}{s + \bar{s} \xrightarrow{a}_D t} \qquad \text{(Choice)}$$

$$\bar{s} + s \xrightarrow{a}_D t$$

Note that there is no transition for **fail**.

Preparatory to the definitions of $\mathcal{O}$ and $\mathcal{D}$ for L_4, we first introduce the operator of reduction, denoted by *red*, from $\mathbb{S}$ to $\mathbb{S}$. Informally speaking, for each $X \in \mathbb{S}$, $red(X)$ delivers the result of applying all possible 'simplifications' $\{\delta\} \cup Y = Y$ (for $Y \neq \emptyset$) in X. In the formal definition of $red(X)$ we use the auxiliary notation (for any $a \in A$, $Y \in \mathbb{S}$) $Y_a =^{def} \{v | a.v \in Y \text{ and } v \neq \epsilon\}$. Note that Y_a may be empty.

DEFINITION 6.5. $red(\{\delta\}) = \{\delta\}$,

$$red(X) = \{a | a \in X\} \cup \bigcup \{a.red(X_a) | a \in A \text{ and } X_a \neq \emptyset\}, \text{ if } X \neq \{\delta\}$$

Well-definedness of *red* follows as usual.

The operational semantics, collecting successive steps in a way which is an adaptation of the one used previously, is given in

DEFINITION 6.6.

a. $\mathcal{O}: \mathcal{P}rog_4 \to \mathbb{S}$ is given by $\mathcal{O}[\![<D|s>]\!] = \mathcal{O}_D[\![s]\!]$.

b. $\mathcal{O}_D: L_4 \to \mathbb{S}$ is given by

$$\mathcal{O}_D[\![s]\!] = red(\{a | s \xrightarrow{a}_D E\} + \bigcup \{a.\mathcal{O}_D[\![s]\!] | s \xrightarrow{a}_D s'\}) \quad \text{the argument of } red \text{ is nonempty}$$

$$= \{\delta\}, \quad \text{otherwise.}$$

Well-definedness of $\mathcal{O}_D$ is established by the usual contractivity argument.

For the denotational semantics for L_4, we first have to define the semantic operators '$\circ$', '$+$', '$\|$' (and the auxiliary operator of *left merge* '$\llcorner$').

DEFINITION 6.7 (the semantic operators $+, \circ, \|, \llcorner$). Let $X, Y \in \mathbb{S}$.

a. $X + Y = red(X \cup Y)$, where '$\cup$' is the set-theoretic union of elements in $\mathbb{S}$.

b. Let the operator $\Phi_\circ: (\mathbb{S} \times \mathbb{S} \to {}^1\mathbb{S}) \to (\mathbb{S} \times \mathbb{S} \to {}^1\mathbb{S})$ be defined as follows: Let $\phi \in \mathbb{S} \times \mathbb{S} \to {}^1\mathbb{S}$, and let us write $\tilde{\phi}_\circ$ for $\Phi_\circ(\phi)$. We put

$$\tilde{\phi}_\circ(\{\delta\})(Y) = \{\delta\}$$

$$\tilde{\phi}_\circ(X)(Y) = \bigcup \{a.\phi(X_a)(Y) | a \in A \text{ and } X_a \neq \emptyset\} + \bigcup \{a.Y | a \in X\} \quad \text{for} \quad X \neq \{\delta\}$$

c. Let the operator $\Phi_{\|}$: $(\mathbb{S}\times\mathbb{S}\to{}^1\mathbb{S})\to(\mathbb{S}\times\mathbb{S}\to{}^1\mathbb{S})$ be defined as follows: let $\phi\in\mathbb{S}\times\mathbb{S}\to{}^1\mathbb{S}$, and let us write $\tilde{\phi}_{\|}$ for $\Phi_{\|}(\phi)$. We put

$$\tilde{\phi}_{\|}(X)(Y) = \tilde{\phi}_{\circ}(X)(Y)+\tilde{\phi}_{\circ}(Y)(X)$$

d. Let $\circ=$ fixed point $(\Phi_{\circ})$, $\|=$ fixed point $(\Phi_{\|})$, $\llcorner=\Phi_{\circ}(\|)$.

LEMMA 6.8. *The above definitions are well-defined. In particular,* $\Phi_{\circ}$: $(\mathbb{S}\times\mathbb{S}\to{}^1\mathbb{S})\to^{\frac{1}{2}}(\mathbb{S}\times\mathbb{S}\to{}^1\mathbb{S})$, *and similarly for* $\Phi_{\|}$. *Also, the operators* $+$, $.$, $\|$, $\llcorner$ *are ndi.*

PROOF. Standard. Apart from minor variations, the required calculations can be found, e.g., in appendix B of [BZ]. □

We proceed with the denotational semantics proper. Let $\Gamma_4=\mathbb{X}\to\mathbb{S}$. The mappings $\mathcal{D}$: $L_4\to\Gamma_4\to\mathbb{S}$ and $\mathcal{M}$: $\mathcal{P}rog_4\to\mathbb{S}$ are given in

DEFINITION 6.9.

a. $\mathcal{D}[\![a]\!]\gamma=\{a\}$, $\mathcal{D}[\![x]\!]\gamma=\gamma x$, $\mathcal{D}[\![\mathbf{fail}]\!]\gamma=\{\delta\}$

b. $\mathcal{D}[\![s_1\ \mathbf{op}\ s_2]\!]\gamma=\mathcal{D}[\![s_1]\!]\gamma\ op\ \mathcal{D}[\![s_2]\!]\gamma$, where **op** ranges over the syntactic operators ;, $\|$, $+$ and *op* ranges or the semantic operators $\circ$, $\|$, $+$, respectively.

c. $\mathcal{M}[\![<D|s>]\!]=\mathcal{D}[\![s]\!]\gamma_D$, where, for $D\equiv<x_i\Leftarrow g_i>_i$, we put (as usual) $\gamma_D=\gamma\{X_i/x_i\}_i$, and

$$<X_1,\cdots,X_n> = \text{fixed point } <\Psi_1,\cdots,\Psi_n>$$

with $\Psi_j=\lambda Y_1.\cdots.\lambda Y_n.\mathcal{D}[\![g_j]\!]\gamma\{Y_i/x_i\}_i$.

We have the usual equivalence theorem

THEOREM 6.10. $\mathcal{O}[\![\sigma]\!]=\mathcal{M}[\![\sigma]\!]$, *for all* $\sigma\in\mathcal{P}rog_4$.

PROOF. Let Ψ_D: $(L_4\to\mathbb{S})\to(L_4\to\mathbb{S})$ be defined as follows. Take any $F\in L_4\to\mathbb{S}$. We put

$$\Psi_D(F)(s) = \{a|s\overset{a}{\to}_D E\}+\bigcup\{a.F(s')|s\overset{a}{\to}_D s'\} \quad \text{the argument of } red \text{ is nonempty}$$

$$= \{\delta\}, \quad \text{otherwise.}$$

Let $\mathcal{D}_D$: $L_4\to\mathbb{S}$ be defined by $\mathcal{D}_D[\![s]\!]=\mathcal{D}[\![s]\!]\gamma_D$. We shall show that $(\star)$ $\Psi_D(\mathcal{D}_D)=\mathcal{D}_D$, thus establishing that $\mathcal{D}_D=\mathcal{O}_D$ and, hence $\mathcal{O}=\mathcal{M}$, by the usual argument.

STAGE 1. First we prove that $\Psi_D(\mathcal{D}_D)(g)=\mathcal{D}_D[\![g]\!]$ for each $g\in L_4^{\delta}$, using induction on the complexity of g. The cases $g\equiv a$ or $g\equiv\mathbf{fail}$ are clear. For the other cases, we first observe that it is easily verified (by an inductive argument on the complexity of g) that g has no transitions $g\overset{a}{\to}_D\cdots$ iff $\mathcal{D}_D[\![g]\!]=\{\delta\}$. (Note the g has no transitions iff g obeys the syntax $g::=\mathbf{fail}|g;s|g_1\|g_2|g_1+g_2$.) We now consider the case that g has indeed transitions. We then have

$g\equiv g_1;s$.

$$\Psi_D(\mathcal{D}_D)(g_1;s) = \{a|g_1;s\overset{a}{\to}E\}+\bigcup\{a.\mathcal{D}_D[\![\bar{s}]\!]|g_1;s\overset{a}{\to}_D\bar{s}\}=$$

$$(\{a|g_1\overset{a}{\to}_D E\}+\bigcup\{a.\mathcal{D}_D[\![s']\!]|g_1\overset{a}{\to}_D s'\})\circ\mathcal{D}_D[\![s]\!]=\Psi_D(\mathcal{D}_D)(g_1)\circ\mathcal{D}_D[\![s]\!]=$$

$$\text{(ind. hyp.) } \mathcal{D}_D[\![g_1]\!]\circ\mathcal{D}_D[\![s]\!] = \mathcal{D}_D[\![g_1;s]\!]$$

$g\equiv g_1\|g_2$.

$$\Psi_D(\mathcal{D}_D)(g_1\|g_2) = \{a|g_1\|g_2\overset{a}{\to}_D E\}+\bigcup\{a.\mathcal{D}_D[\![\bar{s}]\!]|g_1\|g_2\overset{a}{\to}_D\bar{s}\}=$$

$$\bigcup\{a.\mathcal{D}_D[\![g_2]\!] \mid g_1 \xrightarrow{a}_D E\} + \bigcup\{a.\mathcal{D}_D[\![s' \| g_2]\!] \mid g_1 \xrightarrow{a}_D s'\} +$$

$$\bigcup\{a.\mathcal{D}_D[\![g_1]\!] \mid g_2 \xrightarrow{a}_D E\} + \bigcup\{a.\mathcal{D}_D[\![g_1 \| s'']\!] \mid g_2 \xrightarrow{a}_D s''\} =$$

$$(\bigcup\{a \mid g_1 \xrightarrow{a}_D E\} + \bigcup\{a.\mathcal{D}_D[\![s']\!] \mid g_1 \xrightarrow{a}_D s'\}) \mathbb{L} \mathcal{D}_D[\![g_2]\!] +$$

$$(\bigcup\{a \mid g_2 \xrightarrow{a}_D E\} + \bigcup\{a.\mathcal{D}_D[\![s'']\!] \mid g_2 \xrightarrow{a}_D s''\}) \mathbb{L} \mathcal{D}_D[\![g_1]\!] +$$

$$(\Psi_D(\mathcal{D}_D)(g_1) \mathbb{L} \mathcal{D}_D[\![g_2]\!]) + (\Psi_D(\mathcal{D}_D)(g_2) \mathbb{L} \mathcal{D}_D[\![g_1]\!]) = \text{(twice the ind. hyp.)}$$

$$(\mathcal{D}_D[\![g_1]\!] \mathbb{L} \mathcal{D}_D[\![g_2]\!]) + (\mathcal{D}_D[\![g_2]\!] \mathbb{L} \mathcal{D}_D[\![g_1]\!]) = \mathcal{D}_D[\![g_1 \| g_2]\!].$$

$g \equiv g_1 + g_2$. Left to the reader.

STAGE 2. We now prove that $\Psi_D(\mathcal{D}_D)(s) = \mathcal{D}_D[\![s]\!]$, for all $s \in L_4$, by induction on the complexity of s. All cases are as in stage 1, but for the case $s \equiv x$. We only consider the subcase that $\Psi_D(\mathcal{D}_D)(x) \neq \{\delta\}$. We have $\Psi_D(\mathcal{D}_D)(x) = \{a \mid x \xrightarrow{a}_D E\} + \bigcup\{a.\mathcal{D}_D[\![\bar{s}]\!] \mid x \xrightarrow{a}_D \bar{s}\} =$
$\{a \mid g \xrightarrow{a}_D E\} + \bigcup\{a.\mathcal{D}_D[\![\bar{s}]\!] \mid g \xrightarrow{a}_D \bar{s}\} = \Psi_D(\mathcal{D}_D)(g)$ (with $x \Leftarrow g$ in D) = (stage 1)
$\mathcal{D}_D[\![g]\!] =$ (def. $\mathcal{D}_D$) $\mathcal{D}_D[\![x]\!]$. □

7. (AND/OR) PARALLEL LOGIC PROGRAMMING WITH COMMIT: THE BRANCHING TIME CASE

In the next language studied (L_5) we replace the (noncommitting) sequential operator ';' by the commit ':'. We recall that the essential difference between L_4 and L_5 consists in the fact that, in L_4, we have the equivalence (*) $(s;s_1)+(s;s_2) = s;(s_1+s_2)$. In particular, we have

$$(a; \mathbf{fail}) + (a;b) = a:(\mathbf{fail} + b) = a;b$$

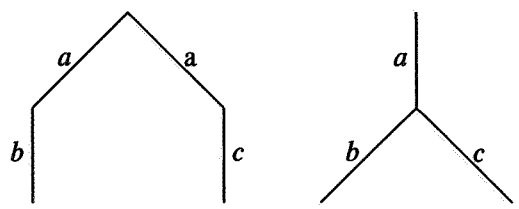
On the other hand, in L_5 we do not have, in general, that (*) holds. In particular, we have that

$$(a: \mathbf{fail}) + (a:b) \neq a:(\mathbf{fail} + b) \quad (= a:b)$$

Our task is, therefore, to develop an underlying mathematical structure which makes sufficient distinctions not to identify (the meaning of) the two L_5-statements $(a: \mathbf{fail}) + (a:b)$ and $a:(\mathbf{fail} + b)$. For this purpose, we use the (metric) process theory as first described in [BZ] (and further elaborated in [AR]), and sketched briefly in Section 2.3. We introduce the domain $(p,q \in) P$ of processes as solution to the equation (isometry, to be precise)

$$P \cong \mathcal{P}_{\text{closed}}(A \cup (A \times P)) \tag{7.1}$$

Elements in P are, for example, $p_1 = \{a\}$, $p_2 = \{<a, \{b\}>\}$, $p_3 = \{<a, \{b\}>, <a, \{c\}>\}$, $p_4 = \{<a, \{b,c\}>\}$, $p_5 = \{<a, \varnothing>\}$, $p_6 = \{<a, \{<a, \{<a, \cdots>\}>\}>\}$, $p_7 = \{a, <a, \{a\}>, <a, \{<a, \{a\}>\}>, \cdots\}$. We observe, for example, that p_3 and p_4 are different processes. In a picture, we can represent them as



respectively. Process p_6 can be obtained as $\lim_n p_n'$, with p_0' arbitrary, $p'_{n+1} = \{<a, p'_n>\}$. Process p_7 equals $\lim_n p''_n$, with p''_0 arbitrary, $p''_{n+1} = \{a\} \cup (p''_n : \{a\})$ (see below for the operators $\cup$, : on processes). We emphasize that the empty set $\varnothing$ is a process (in (7.1) we use $\mathcal{P}_{\text{closed}}(\cdot)$ rather than

$\mathcal{P}_{\text{nonempty closed}}(\cdot)$). The empty process has indeed the appropriate properties to model failure (note that δ has disappeared from the scence in Section 7): We shall subsequently define the semantic operators '$\cup$' and ':' such that $\emptyset \cup p = p \cup \emptyset = p$, $\emptyset : p = \emptyset$, but $p : \emptyset \neq \emptyset$ (in general).

After this introduction, we first give the syntax for L_5:

DEFINITION 7.1.

a (statements). The class of statements $(s \in) L_5$ is defined by

$$s ::= a | x | \textbf{fail} | s_1 : s_2 | s_1 \| s_2 | s_1 + s_2$$

b. The classes $L\xi$, $\mathcal{D}ecl_5$ and $\mathcal{P}rog_5$ are defined as usual.

REMARK. The reader who would like to see constructs $:s$ or $s:$ (commit with empty left- or right-operand) will have to take the trouble to incorporate a silent atomic action τ in A, and read $\tau : s$ for $:s$, $s : \tau$ for $s:$.

For the definition of the operational semantics for L_5, we introduce the transition system T_5. Fortunately, only one minor variation in the system T_4 is required. We replace the rule for (Seq Comp) by

$$\frac{s \xrightarrow{a}_D s' | E}{s : \bar{s} \xrightarrow{a}_D s' : \bar{s} | \bar{s}} \qquad \text{(Commit)}$$

and keep all other rules of T_4 unchanged (including the rules for (Par Comp)).

The essential new element in the operational semantics for L_5 is the way in which the individual transitions, based on T_5, are assembled together to form a process $p \in P$ (rather than a set $X \in \mathfrak{S}$ as was the case for L_4). This is described in

DEFINITION 7.2.

a. $\mathcal{O} : \mathcal{P}rog_5 \to P$ is given by $\mathcal{O}[\![< D | s >]\!] = \mathcal{O}_D[\![s]\!]$.

b. $\mathcal{O}_D[\![s]\!] = \{a | s \xrightarrow{a}_D E\} \cup \{< a, \mathcal{O}_D[\![s']\!] > | s \xrightarrow{a}_D s'\}$.

Comparing this definition with Definition 6.5, we see the essential difference in the clause $\cdots \{< a, \mathcal{O}_D[\![s']\!] > | \cdots\}$ which replaces $\cdots \bigcup \{a . \mathcal{O}_D[\![s']\!] | \cdots\}$. In addition, there is no special treatment for the case that the right-hand size in clause b is empty, since the empty process $\emptyset$ is a valid outcome requiring no amendments (in the form of some $\{\delta\}$).

EXAMPLES.

1. $\mathcal{D}_D[\![(a:b) + (a:c)]\!] = \{< a, \{b\} >, < a, \{c\} >\}$
2. $\mathcal{D}_D[\![a : (b + c)]\!] = \{< a, \{b, c\} >\}$
3. $\mathcal{D}_D[\![a + \textbf{fail}]\!] = \{a\}$
4. $\mathcal{D}[\![< x \Leftarrow (a:x) + b | x >]\!] = p$, where $p = \lim_n p_n$, p_0 arbitrary, $p_{n+1} = \{< a, p_n >, b\}$.

For the denotational semantics, we define the operators $+$, $.$, $\|$ (and $\lfloor\!\lfloor$) on processes p, q in P. We follow the pattern of definition as in Definition 6.7. Note, however, that in the present context there is no need for the reduction operation.

DEFINITION 7.3. Let $p, q \in P$.

a. $p \cup q$ is the set theoretic union of (the sets) p, q

b. Let the operator $\Psi_: : (P \times P \to {}^1P) \to (P \times P \to {}^1P)$ be defined as follows. Take $\phi \in P \times P \to {}^1P$.

$$\Psi_:(\phi)(p)(q) = \{< a, q > | a \in p\} \cup \{< a, \phi(p')(q) > | < a, p' > \in p\}$$

c. Let the operator $\Psi_{\|}$: $(P\times P\to^1 P)\to(P\times P\to^1 P)$ be defined as follows. Take $\phi\in P\times P\to^1 P$.

$$\Psi_{\|}(\phi)(p)(q) = \Psi_{:}(\phi)(p)(q)\cup\Psi_{:}(\phi)(q)(p)$$

d. Let $:=$ fixed point $(\Psi_{:})$, $\|=$ fixed point $(\Psi_{\|})$, $\|\_=\Psi_{:}(\|)$.

As before, $\cup$, :, $\|$, $\|\_$ are well-defined and *ndi*.

Let $\Gamma_5=\mathfrak{X}\to P$. The mappings $\mathfrak{D}$: $L_5\to\Gamma_5\to P$ and $\mathfrak{M}$: $Prog_5\to P$ are given in

DEFINITION 7.4.

a. $\mathfrak{D}[\![a]\!]\gamma=\{a\}$, $\mathfrak{D}[\![x]\!]\gamma=\gamma x$, $\mathfrak{D}[\![\mathbf{fail}]\!]\gamma=\varnothing$

b. $\mathfrak{D}[\![s_1\ \mathbf{op}\ s_2]\!]\gamma=\mathfrak{D}[\![s_1]\!]\gamma\ op\ \mathfrak{D}[\![s_2]\!]\gamma$, where **op** ranges over the syntactic operators $+$, :, $\|$, and *op* over the semantic operators $\cup$, :, $\|$, respectively.

c. $\mathfrak{M}[\![<D|s>]\!]=\mathfrak{D}[\![s]\!]\gamma_D$, with γ_D as usual.

The equivalence of Θ and $\mathfrak{M}$ for $\mathscr{Prog}_5$ is established in almost the same way as this was done for $\mathscr{Prog}_4$. In fact, the only difference is that in the present case the proof is slightly simpler, since the complications having to do with the reduction operator have disappeared. Thus, we have

THEOREM 7.5. *For each* $\sigma\in\mathscr{Prog}_5$, $\Theta[\![\sigma]\!]=\mathfrak{M}[\![\sigma]\!]$.

By way of conclusion of this section we observe that the way our definitions are organized have as remarkable benefit that the definitions of Θ and $\mathfrak{M}$, and the proof of their equivalence, are almost identical for $\mathscr{Prog}_4$ and $\mathscr{Prog}_5$, notwithstanding the essential difference in the underlying mathematical structure: the objects in $\mathbb{S}$ are very much simpler than the objects in P.

8. (AND/OR) PARALLEL LOGIC PROGRAMMING WITH COMMIT: INCREASING THE GRAIN SIZE

The last language, L_6, of our list of abstractions of logic programming languages embodies a version of (and/or) parallel logic programming which combines both the (noncommitting) sequential composition (;) and the commit (:) operator. This language was designed as a step on the way towards the semantic modelling of logic languages such as Concurrent Prolog (CP from [Sh1]). The emphasis is here on CP's commit operator, see [BK] for a discussion of its notion of read-only variables. We do not want to go into details here (once more referring to [BK]). Rather, we give a brief hint as to how CP's constituent concepts appear in L_6. Take a CP program with clauses

head ← *guard* | *body*

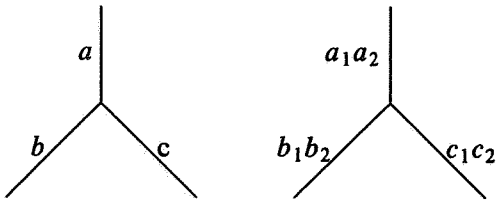
Here *head* is some (logical) atom, *guard* and *body* are conjunctions of (logical) atoms, and | is CP's commit. Such a clause would induce, in a corresponding L_6 program, a declaration of the form

$x\Leftarrow$ (unification step; parallel execution of atoms in *guard*):

(parallel execution of atoms in *body*)

From this declaration (cf. also the appendix), it should at least be clear that the combined presence in L_6 of ';' and ':' is necessary to model normal sequencing together with committing behaviour. (In [BK] another operator is introduced which turns some statement s into an atomic—noninterruptible—version, denoted by [s].)

Why the 'increase in grain size'? Consider, by way of example, a statement such as $s_1\equiv a:(b+c)$ which we compare with $s_2\equiv(a_1;a_2)$: $((b_1;b_2)+(c_1;c_2))$. We shall design our semantic model such that the meanings of s_1 and s_2 are (pictorially) represented by



Thus, the atoms or grains a,b,c are enlarged to a_1a_2, b_1b_2, c_1c_2. In the precise mathematical notation, we obtain the processes $\{<a,\{b,c\}>\}$ and $\{<a_1a_2,\{<b_1b_2, c_1c_2\}>\}$. The presence of entities of the latter form necessitates an extension of the process domain as introduced in Section 7. Instead of P satisfying $P \cong \mathcal{P}_c(A \cup (A \times P))$ we now work with Q satisfying $Q \cong \mathcal{P}_{nc}(R \cup (A^+ \times Q))$. Details follow. Furthermore, the combined presence of ';' and ':' requires a refinement of the transition system T_6 which now involves two types of transitions $\overset{a}{\rightarrow}_1$ and $\overset{a}{\rightarrow}_2$, corresponding to steps of a noncommitting $(a.\ \cdots)$ versus steps of a committing $(<a,\ \cdots>)$ kind.

After these introductory remarks, we are now ready for the precise definitions:
We start with the syntax.

DEFINITION 8.1.
a (statements). The class of statements $(s \in)L_6$ is defined by

$$s ::= a|x|\mathbf{fail}|s_1;s_2|s_1:s_2|s_1\|s_2|s_1+s_2$$

b. The syntactic classes L_6^g, $\mathcal{D}ecl_6$, $\mathcal{P}rog_6$ are obtained from L_6 as usual.

The operational semantics for L_6 is defined in terms of a transition system T_6 and associated definition of θ which provides a synthesis of the ideas for T_4 and T_5 (and their associated definitions of θ).

We first discuss the process domain $(p,q \in)Q$ which we use as semantic universe. Q may be seen as a domain incorporating notions both from the linear time model $\mathbb{S} = \mathcal{P}_{nc}(R)$ (with special role of δ; R as in Section 6) and the branching time model P. We define Q as solution to the isometry

$$Q \cong \mathcal{P}_{\text{nonempty closed}}(R \cup (A^+ \times Q)) \tag{8.1}$$

In a moment, we shall describe formally how a domain Q satisfying (8.1) can be obtained. Informally, we add the following comments. First, note that we do not include $\varnothing$ as a valid element in Q. This is motivated by the reappearance of $\delta \in R$ which, as before, plays the role of modelling failure. Secondly, we consider a few examples of processes q in Q: $q_1 = \{<ab,\{c\}>\}$, $q_2 = \{a^\omega\}$, $q_3 = \{<ab,\{cd, e\}>\}$, $q_4 = \{a\delta, b^\omega\} = \lim_n q'_n$, where q_0' is arbitrary, $q'_{n+1} = \{a\delta, <b^n,\{c\}>\}$. Thirdly, we note that entities $\{<v, q>\}$ are processes for $v \in A^+$, but not for $v \in A^\omega$ or $v \in A^*.\{\delta\}$. Intuitively, it makes no sense to 'perform' q after some v which is infinite or ends in δ. Altogether, we observe a certain interplay between linear time objects intermingled with branching structure. The definitions below will be organized such that ';' influences the linear time aspects. Thus, the semantic operator '$\circ$' modelling ';' will yield, for example, $\{ab\} \circ \{<cd, q>\} = \{<abcd, q>\}$. On the other hand, the commit operator will impose branching structure. E.g., $\{ab\}: \{c, d\} = \{<ab,\{c, d\}>\}$.

The way in which we solve (8.1) does not completely follow the usual pattern of solving domain equations as described in [BZ] or [AR]. Rather, we apply a somewhat more adhoc technique which is a uniform variant of the definitions in [Ko]. We construct a sequence of complete ultrametric spaces $(Q_n, d_n)_n$, defined in the following way:

$$(Q_0, d_0) = (\mathcal{P}_{nc}(R), \hat{d})$$

with $(\mathbb{S} =)\mathcal{P}_{nc}(R)$ and $\hat{d}$ the usual metric on $\mathbb{S}$. Furthermore,

$$Q_{n+1} = \mathcal{P}_{nc}(R \cup (A^+ \times Q_n))$$

where $\mathcal{P}_{nc}(.)$ abbreviates $\mathcal{P}_{\text{nonempty closed}}(.)$, and d_{n+1} is defined as follows: The distance d_{n+1} is the Hausdorff metric (on sets in Q_{n+1}) derived from the point metric $\tilde{d}_{n+1}$ (on elements in $R \cup (A^+ \times Q_n)$) defined in

$$\tilde{d}_{n+1}(v,w) = d(v,w), \quad \text{for} \quad v,w \in R$$

$$\tilde{d}_{n+1}(v, <w,q>) = d(v,w) \quad \text{if} \quad v \neq w$$

$$= 2^{-n}, \quad \text{if} \quad v = w \quad \text{and length } (v) = n$$

$$\tilde{d}_{n+1}(<v,p>, w) \quad \text{similar}$$

$$\tilde{d}_{n+1}(<v,p>, <w,q>) = d(v,w) \quad \text{if} \quad v \neq w$$

$$= 2^{-n}, d_n(p,q), \quad \text{if} \quad v = w \quad \text{and length } (v) = n$$

Observe that $Q_n \subseteq Q_{n+1}$, $n = 0,1, \cdots$. Now let $(Q_\omega, d_\omega) = (\bigcup_n Q_n, \bigcup_n d_n)$, where, for any $p,q \in Q_\omega$, $d_\omega(p,q) = d_m(p,q)$, with $m = \min\{k | p,q \in Q_k\}$. Next, we define (Q, d) as the *completion* of (Q_ω, d_ω). By techniques as in [BZ], it can be shown that (Q, d) satisfies the isometry (8.1).

REMARK. By way of example, note that, by the above definitions, we have that $\lim_n \{a\delta, <b^n, \{c\}>\} = \{a\delta, b^\omega\}$.

NOTATION. For $q \in Q$, we shall use y to range over (the set) q. Thus, y is element of R or of $A^+ \times Q$.

Below, we shall need various semantic operators involving $q \in Q$. The first of these is prefixing a finite nonempty word v to some q:

DEFINITION 8.2. Let $v \in A^+$, $q \in Q$. We put

$$v.q = \{v.y | y \in q\}$$

where $v.w$ is as usual for $w \in R$, and $v.<w,q'> = <v.w,q'>$.

We are now sufficiently prepared for the definition of T_6 and associated $\mathcal{O}$. In the present section, we shall use transitions of three forms

$$s \xrightarrow{a}_D E, \quad s \xrightarrow{a}_{1,D} s', \quad s \xrightarrow{a}_{2,D} s'$$

(From now on, we drop the subscript D for easier readability.) Transitions $s \xrightarrow{a}_1 s'$ are intended to model noncommitting sequential steps - which in the associated definition of $\mathcal{O}$ will reappear as $a.\mathcal{O}[\![s']\!]$. On the other hand, transitions $s \xrightarrow{a}_2 s'$ model commit steps which we find back in the definition of $\mathcal{O}$ as $<a, \mathcal{O}[\![s']\!]>$. Thus, we see the combined appearance of features from Sections 6 and 7. Moreover, the semantic definitions will be organized such that, on the one hand, $(v.p_1) \between p_2 = v.(p_1 \between p_2)$, and, on the other hand, $\{<v,p_1>\} \between p_2 = \{<v, p_1 \| p_2>\}$. (More about this after the definition of T_6.) In order to have the operational semantics respect these identities, the transitions for parallel composition are phrased in terms of '$\between$' rather than of '$\|$' (as before). As last introductory remark we already announce that we shall devote the next section to the analysis of a related system where the transitions $s \xrightarrow{a} E$, $s \xrightarrow{a}_1 s'$, $s \xrightarrow{a}_2 s'$ are replaced by transitions with a larger grain size: instead of $a \in A$ we shall allow arbitrary $v \in A^+$ as 'atomic' steps, and we shall design T_7 in terms of $s \xrightarrow{v} E$, $s \xrightarrow{v}_1 s'$, $s \xrightarrow{v}_2 s'$. To avoid confusion, we emphasize that in the present section we have already increased the grain size *of the processes,* working with processes such as

$\{<ab,\{<cd,\{e,f\}>\}>\}$ instead of (only) with processes such as $\{<a,\{<b,\{<c,\{e,f\}>\}>\}>\}$.

We present the system T_6 for L_6. We shall also use the notation $s \xrightarrow{a}_i s'$ as shorthand for any of the three possibilities $s \xrightarrow{a} E$, $s \xrightarrow{a}_1 s'$, $s \xrightarrow{a}_2 s'$.

DEFINITION 8.3 (transition system T_6).

$$a \xrightarrow{a} E \qquad \text{(Elem)}$$

$$\frac{g \xrightarrow{a}_i s}{x \xrightarrow{a}_i s}, \quad x \Leftarrow g \quad \text{in} \quad D \qquad \text{(Rec)}$$

$$\frac{s \xrightarrow{a}_i s'}{s + \bar{s} \xrightarrow{a}_i s' \qquad \bar{s} + s \xrightarrow{a}_i s'} \qquad \text{(Choice)}$$

$$\frac{s_1 \lfloor\!\lfloor s_2 \xrightarrow{a}_i s'}{s_1 \| s_2 \xrightarrow{a}_i s' \qquad s_2 \| s_1 \xrightarrow{a}_i s'} \qquad \text{(Par Comp)}$$

$$\frac{s \xrightarrow{a} E}{s ; \bar{s} \xrightarrow{a}_1 \bar{s}} \qquad (\rightarrow_1 \text{ intro})$$

$$\frac{s \xrightarrow{a} E}{s : \bar{s} \xrightarrow{a}_2 \bar{s} \qquad s \lfloor\!\lfloor \bar{s} \xrightarrow{a}_2 \bar{s}} \qquad (\rightarrow_2 \text{ intro})$$

$\dfrac{s \xrightarrow{a}_1 s'}{s ; \bar{s} \xrightarrow{a}_1 s' ; \bar{s}}$	$\dfrac{s \xrightarrow{a}_2 s'}{s ; \bar{s} \xrightarrow{a}_2 s' ; \bar{s}}$	(Seq Comp)
$s : \bar{s} \xrightarrow{a}_1 s' : \bar{s}$	$s : \bar{s} \xrightarrow{a}_2 s' : \bar{s}$	(Commit)
$s \lfloor\!\lfloor \bar{s} \xrightarrow{a}_1 s' \lfloor\!\lfloor \bar{s}$	$s \lfloor\!\lfloor \bar{s} \xrightarrow{a}_2 s' \| \bar{s}$	(Left Merge)

The axiom and first two rules of T_6 are clear. The rule for (Par Comp) states that a step from $s_1 \| s_2$ is either a step from s_1 (case $s_1 \lfloor\!\lfloor s_2$) or from s_2 (case $s_2 \lfloor\!\lfloor s_1$). The next two rules introduce the $\rightarrow_1$ and $\rightarrow_2$ transitions. In the final group of rules, the type of transition ($\rightarrow_1$ or $\rightarrow_2$) is always inherited. We draw in particular attention to the rules for left merge. After performing an $\xrightarrow{a}_1$ step from $s \lfloor\!\lfloor \bar{s}$, the step after that has again to be from the (new) left operand in $s' \lfloor\!\lfloor \bar{s}$. On the other hand, after taking an $\xrightarrow{a}_2$ step, next a step from both operands (in $s' \| \bar{s}$) is possible.

Before defining $\mathfrak{O}$ and $\mathfrak{P}$ for L_6, due to the reappearance of δ, we again have to *reduce* processes by applying, wherever possible, simplifications $\{\delta\} \cup p = p$ (note that p is now nonempty by the definition of Q). Reduction is defined in

DEFINITION 8.4.
a. For $p \in Q$, $v \in A^+$ we put

$$p_v = \{y | v.y \in p\}$$

b. We define the mapping *red*: $Q \to Q$ by $red(\{\delta\}) = \{\delta\}$. For $p \neq \{\delta\}$, we put

$$red(p) = \{a | a \in p\} \cup \bigcup \{v.red(p_v) | p_v \neq \varnothing\} \cup \{<a, red(p')> | <a, p'> \in p\}$$

REMARK. Note that, in clause *a*, p_v may be the empty set and that, since y ranges over $R \cup (A^+ \times Q)$, y cannot be ϵ in the definition of p_v.

EXAMPLES. $red(\{\delta, a\}) = \{a\}$, $red(\{a\delta, ab, c\}) = \{ab, c\}$,
$red(\{a\delta, <ab, \{c\}>\}) = a.red(\{\delta, <b, \{c\}>\}) = \{<ab, \{c\}>\}$.
We can now give

DEFINITION 8.5.
a. $\Theta: \mathscr{P}rog_6 \to Q$ is defined as $\Theta[\![<D|s>]\!] = \Theta_D[\![s]\!]$.
b. $\Theta_D[\![s]\!] = red\ (\{a | s \xrightarrow{a} E\} \cup \bigcup \{a.\Theta_D[\![s']\!] | s \xrightarrow{a}_1 s'\} \cup \{<a, \Theta_D[\![s']\!]> | s \xrightarrow{a}_2 s'\})$

if the argument of *red* is nonempty
$= \{\delta\}$, otherwise.

We see the already discussed mixed character of the right-hand side delivering both noncommitting and committing outcomes ($a. \cdots$ and $<a, \cdots>$).

Θ_D is well-defined by the familiar contractivity argument. It may be enlightening to observe that the presence of an empty process $\varnothing$ in our domain would invalidate this argument. Allowing p or q to be empty we no longer have that $d(a.p, a.q) \leqslant \frac{1}{2} d(p,q)$, a property which does hold for nonempty p and q. Note that in both scenarios (with or without empty processes), we have that $d(<a,p>, <a,q>) = \frac{1}{2} d(p,q)$.

We proceed with the denotational definitions. The definition of the various semantic operators is now somewhat more involved. The operators $+, \circ, :, \|, \llcorner$ are defined in

DEFINITION 8.6. Let $p, q \in Q$
a. $p + q = red(p \cup q)$, where '$\cup$' is the set-theoretic union of (the sets) p, q
b. The higher order mappings $\Phi_\circ$, $\Phi_:$, $\Phi_\|$, all from $(Q \times Q \to^1 Q)$ to $(Q \times Q \to^1 Q)$ are defined as follows. Let $\phi \in Q \times Q \to^1 Q$.

$$\Phi_\circ(\phi)(p)(q) = \{v.q | v \in p \cap A^+\} + \{v | v \in p \cap (A^\omega \cup A^*.\{\delta\})\} + \{<v, \phi(p')(q)> | <v, p'> \in p\}$$

$$\Phi_:(\phi)(p)(q) = \{<v, q> | v \in p \cap A^+\} + \{v | v \in p \cap (A^\omega \cup A^*.\{\delta\})\} + \{<v, \phi(p')(q)> | <v, p'> \in p\}$$

$$\Phi_\|(\phi)(p)(q) = \Phi_:(\phi)(p)(q) + \Phi_:(\phi)(q)(p)$$

c. We put $\circ =$ fixed point $(\Phi_\circ)$, $: =$ fixed point $(\Phi_:)$, $\| =$ fixed point $(\Phi_\|)$, $\llcorner = \Phi_:(\|)$.

The definitions of $\mathfrak{D}$ and $\mathfrak{M}$ are now standard. Let $\Gamma_6 = \mathscr{K} \to Q$. We define $\mathfrak{D}: L_6 \to \Gamma_6 \to Q$ and $\mathfrak{M}: \mathscr{P}rog_6 \to Q$ in

DEFINITION 8.7.
a. $\mathfrak{D}[\![a]\!]\gamma = \{a\}$, $\mathfrak{D}[\![x]\!]\gamma = \gamma x$, $\mathfrak{D}[\![\mathbf{fail}]\!]\gamma = \{\delta\}$, $\mathfrak{D}[\![s_1 \mathbf{op} s_2]\!]\gamma = \mathfrak{D}[\![s_1]\!]\gamma\ op\ \mathfrak{D}[\![s_2]\!]\gamma$, with **op** ranging over $;, :, \|, +$, *op* ranging over $\circ, :, \|, +$, respectively
b. $\mathfrak{M}[\![<D|s>]\!] = \mathfrak{D}[\![s]\!]\gamma_D$, with γ_D as usual.

We conclude this section with the proof of

THEOREM 8.8. *For $\mathcal{O}_D$, $\mathcal{D}$, γ_D as before, and $s \in L_6$:*

$$\mathcal{O}_D[\![s]\!] = \mathcal{D}[\![s]\!]\gamma_D$$

PROOF. Let $\mathcal{O} = \lambda s.\mathcal{D}[\![s]\!]\gamma_D$. As always, it is sufficient to show that $\mathcal{D}_D$ is a fixed point of the operator $\Psi_D: (L_6 \to Q) \to (L_6 \to Q)$ given (for $F \in L_6 \to Q$) by

$$\Psi_D(F)(s) = red(\{a | s \xrightarrow{a} E\} \cup \bigcup \{a.F(s') | s \xrightarrow{a}_1 s'\} \cup \{<a, F(s')> | s \xrightarrow{a}_2 s'\})$$

if the argument of *red* is nonempty

$$= \{\delta\}, \quad \text{otherwise.}$$

The proof follows the pattern as in the proof of Theorem 6.10. Essential intermediate results are the following:

$$\Psi_D(\mathcal{D}_D)(g;s) = \Psi_D(\mathcal{D}_D)(g) \circ \mathcal{D}_D[\![s]\!]$$

(this uses that $(a.p) \circ q = a.(p \circ q)$ and $\{<a, p>\} \circ q = \{<a, p \circ q>\}$)

$$\Psi_D(\mathcal{D}_D)(g:s) = \Psi_D(\mathcal{D}_D)(g) : \mathcal{D}_D[\![s]\!]$$

(this uses that $(a.p) : q = a.(p:q)$ and $\{<a, p>\} : q = \{<a, p:q>\}$)

$$\Psi_D(\mathcal{D}_D)(g_1 + g_2) = \Psi_D(\mathcal{D}_D)(g_1) + \Psi_D(\mathcal{D}_D)(g_2)$$

$$\Psi_D(\mathcal{D}_D)(g_1 \| g_2) = \Psi_D(\mathcal{D}_D)(g_1 \mathbin{\mathbb{L}} g_2) + \Psi_D(\mathcal{D}_D)(g_2 \mathbin{\mathbb{L}} g_1)$$

$$\Psi_D(\mathcal{D}_D)(g_1 \mathbin{\mathbb{L}} g_2) = \Psi_D(\mathcal{D}_D)(g_1) \mathbin{\mathbb{L}} \mathcal{D}_D[\![g_2]\!]$$

this uses that $(a.p) \mathbin{\mathbb{L}} q = a.(p \mathbin{\mathbb{L}} q)$ and $\{<a, p>\} \mathbin{\mathbb{L}} q = \{<a, p \| q>\}$)

These results are to be embedded in an argument which is very much like that of the proof of Theorem 6.10. □

9. INCREASING THE GRAIN SIZE IN THE TRANSITIONS

We conclude our study of flow of control concepts in uninterpreted logic programming with the discussion of a somewhat more specialized topic. We ask (and shall answer affirmatively) whether it is also feasible to base $\mathcal{O}$ for L_6 on transitions

$$s \xrightarrow{\nu} E, \quad s \xrightarrow{\nu}_1 s', \quad s \xrightarrow{\nu}_2 s',$$

thus increasing the grain size of the atomic steps. One might defend the case that this is a more natural style of transitions since the semantic domain is designed such that 'steps' ν, w etc. (appearing in the process domain Q) take the place of the 'steps' $a, b, \cdots$ (from the process domain P). In other words, we are now dealing with processes such as $\{<\nu, \{<w, \cdots>\}>\}$ rather than $\{<a, \{<b, \cdots>\}>\}$, explaining why it is natural to also increase the step size in the transitions. We shall indeed demonstrate in this section that, on the basis of a rather natural extension of T_6, we can define an operational semantics (derived from transitions $s \xrightarrow{\nu}_i s'$) which is equivalent to the denotational semantics of Section 8 (Definition 8.7) and, hence, also with $\mathcal{O}$ as in Definition 6.8. The price to be paid for this somewhat more satisfactory operational semantics is rather more effort to be spent on the equivalence proof.

DEFINITION 9.1. The system T_7 consists of

a. The axiom $a \xrightarrow{a} E$

b. All rules of T_6, with a throughout replaced by ν
c. The new rule

$$\frac{s_1 \xrightarrow{\nu_1} E,\ s_2 \xrightarrow{\nu_2}_i s'}{s_1;s_2 \xrightarrow{\nu_1\nu_2}_i s'} \qquad \text{(Inc Atom)}$$

The accompanying definition for $\mathcal{O}^*$ is

DEFINITION 9.2.
a. $\mathcal{O}^*: \mathscr{P}rog_6 \to Q$ is given by $\mathcal{O}^*[\![<D|s>]\!] = \mathcal{O}^*_D[\![s]\!]$
b. $\mathcal{O}^*_D[\![s]\!] = red(\{\nu | s \xrightarrow{\nu} E\} \cup \bigcup \{\nu.\mathcal{O}^*_D[\![s']\!] | s \xrightarrow{\nu}_1 s'\} \cup \{<\nu, \mathcal{O}^*_D[\![s']\!]> | s \xrightarrow{\nu}_2 s'\})$,
if the argument of *red* is nonempty
$= \{\delta\}$, otherwise,
where the transitions are with respect to T_7.

We shall prove, for $\mathfrak{M}$ as in Definition 8.7,

THEOREM 9.3. *For each* $\sigma \in \mathscr{P}rog_6$, $\mathcal{O}^*[\![\sigma]\!] = \mathfrak{M}[\![\sigma]\!]$.

PROOF. We define the usual mapping Ψ_D: $(L_6 \to Q) \to (L_6 \to Q)$. Take $F \in L_6 \to Q$. We put

$$\Psi_D(F)(s) = red(\{\nu | s \xrightarrow{\nu} E\} \cup \bigcup \{\nu.F(s') | s \xrightarrow{\nu}_1 s'\} \cup \{<\nu, F(s')> | s \xrightarrow{\nu}_2 s'\})$$

if the argument of *red* is nonempty

$= \{\delta\}$, otherwise.

We show that $\Psi_D(\mathcal{D}_D) = \mathcal{D}_D$ $(=^{df} \lambda s.\mathcal{D}[\![s]\!]\gamma_D)$. The proof follows the standard pattern, but the rule (Inc Atom) causes some complications. We first state the key properties of $\Psi_D(\mathcal{D}_D)(s)$, for s ranging over L_6.

$s \equiv a$	$\Psi_D(\mathcal{D}_D)(a) = \{a\}$
$s \equiv x$	$\Psi_D(\mathcal{D}_D)(x) = \Psi_D(\mathcal{D}_D)(g)$, $x \Leftarrow g$ in D
$s \equiv$ **fail**	$\Psi_D(\mathcal{D}_D)(\textbf{fail}) = \{\delta\}$
$s \equiv s_1;s_2$	$\Psi_D(\mathcal{D}_D)(s_1;s_2) = \Psi_D(\mathcal{D}_D)(s_1) \circ \mathcal{D}_D[\![s_2]\!] + \{\nu | s_1 \xrightarrow{\nu} E\} \circ \Psi_D(\mathcal{D}_D)(s_2)$
$s \equiv s_1 : s_2$	$\Psi_D(\mathcal{D}_D)(s_1 : s_2) = \Psi_D(\mathcal{D}_D)(s_1) : \mathcal{D}_D[\![s_2]\!]$
$s \equiv s_1 + s_2$	$\Psi_D(\mathcal{D}_D)(s_1 + s_2) = \Psi_D(\mathcal{D}_D)(s_1) + \Psi_D(\mathcal{D}_D)(s_2)$
$s \equiv s_1 \| s_2$	$\Psi_D(\mathcal{D}_D)(s_1 \| s_2) = \Psi_D(\mathcal{D}_D)(s_1 \lfloor\!\lfloor s_2) + \Psi_D(\mathcal{D}_D)(s_2 \lfloor\!\lfloor s_1)$
$s \equiv s_1 \lfloor\!\lfloor s_2$	$\Psi_D(\mathcal{D}_D)(s_1 \lfloor\!\lfloor s_2) = \Psi_D(\mathcal{D}_D)(s_1) \lfloor\!\lfloor \mathcal{D}_D[\![s_2]\!]$

We give some details of the cases $s \equiv s_1;s_2$ and $s \equiv s_1 \lfloor\!\lfloor s_2$. We consider only the cases that the outcomes are $\neq \{\delta\}$.

$$\Psi_D(\mathcal{D}_D)(s_1;s_2) = red(\{\nu | s_1;s_2 \xrightarrow{\nu} E\} \cup \bigcup \{\nu.\mathcal{D}_D[\![\bar{s}]\!] | s_1;s_2 \xrightarrow{\nu}_1 \bar{s}\} \cup \{<\nu, \mathcal{D}_D[\![\bar{s}]\!]> | s_1;s_2 \xrightarrow{\nu}_2 \bar{s}\}) =$$

$$(\{\nu_1 | s_1 \xrightarrow{\nu_1} E\} \circ \mathcal{D}_D[\![s_2]\!] + \bigcup \{\nu_1.\mathcal{D}_D[\![s']\!] | s_1 \xrightarrow{\nu_1}_1 s'\} \circ \mathcal{D}_D[\![s_2]\!] +$$

$$\{<\nu_1, \mathcal{D}_D[\![s']\!]> | s_1 \xrightarrow{\nu_2}_2 s'\} \circ \mathcal{D}_D[\![s_2]\!])$$

$$+ (\{\nu_1\nu_2 | (s_1 \xrightarrow{\nu_1} E) \wedge (s_2 \xrightarrow{\nu_2} E)\} + \bigcup \{(\nu_1\nu_2).\mathcal{D}_D[\![s'']\!] | (s_1 \xrightarrow{\nu_1} E) \wedge (s_2 \xrightarrow{\nu_2}_1 s'')\}$$

$$+\{<\nu_1\nu_2, \mathcal{D}_D[\![s'']\!]>|(s_1 \xrightarrow{\nu_1} E)\wedge(s_2 \xrightarrow{\nu_2}_2 s'')\})$$

$$= \Psi_D(\mathcal{D}_D)(s_1).\mathcal{D}_D[\![s_2]\!]+\{\nu_1|s_1 \xrightarrow{\nu_1} E\}.\Psi_D(\mathcal{D}_D)(s_2)$$

$$\Psi_D(\mathcal{D}_D)(s_1 \mathbb{L} s_2) = \{\nu_1|s_1 \xrightarrow{\nu_1} E\}\mathbb{L}\mathcal{D}_D[\![s_2]\!]+ \bigcup\{\nu_1.\mathcal{D}_D[\![s']\!]|s_1 \xrightarrow{\nu_1}_1 s'\}\mathbb{L}\mathcal{D}_D[\![s_2]\!]+$$

$$\{<\nu_1, \mathcal{D}_D[\![s']\!]>|s_1 \xrightarrow{\nu_1} s'\}\mathbb{L}\mathcal{D}_D[\![s_2]\!] = \Psi_D(\mathcal{D}_D)(s_1)\mathbb{L}\mathcal{D}_D[\![s_2]\!]$$

Next, we show that $\Psi_D(\mathcal{D}_D)=\mathcal{D}_D$, thus establishing that $\mathcal{O}^*_D=\mathcal{D}_D$, whence $\mathcal{O}^*=\mathcal{M}$ on $\mathcal{P}rog_6$.

We abbreviate $\Psi_D(\mathcal{D}_D)$ to $\tilde{\Psi}_D$, and we prove that $d(\mathcal{D}_D, \tilde{\Psi}_D)=0$.

STAGE 1. For each $g\in L^g_6$,

$$d(\mathcal{D}_D[\![g]\!], \tilde{\Psi}_D(g)) \leqslant \tfrac{1}{2}d(\mathcal{D}_D, \tilde{\Psi}_D)$$

(Note that, clearly, for all s, $d(\mathcal{D}_D[\![s]\!], \tilde{\Psi}_D(s))\leqslant d(\mathcal{D}_D, \tilde{\Psi}_D)$.) We use induction on the complexity of g, and treat here only the (most complex) case $g\equiv g_1;s$. We have

$$\mathcal{D}_D[\![g_1;s]\!] = \mathcal{D}_D[\![g_1]\!]\circ\mathcal{D}_D[\![s]\!]= \text{(by Lemma 9.4 below) } (\mathcal{D}_D[\![g_1]\!]\circ\mathcal{D}_D[\![s]\!]+(\{\nu|g_1 \xrightarrow{\nu} E\}\circ\mathcal{D}_D[\![s]\!])$$

$$\tilde{\Psi}_D(g_1;s) = (\tilde{\Psi}_D(g_1)\circ\mathcal{D}_D[\![s]\!])+(\{\nu|g_1 \xrightarrow{\nu_1} E\}\circ\tilde{\Psi}_D(s))$$

Clearly, we have $d(\mathcal{D}_D[\![g_1]\!]\circ\mathcal{D}_D[\![s]\!], \tilde{\Psi}_D(g_1)\circ\mathcal{D}_D[\![s]\!]) \leqslant d(\mathcal{D}_D[\![g_1]\!], \Psi_D(g_1)) \leqslant$ (ind. hyp.) $\frac{1}{2}d(\mathcal{D}_D, \Psi_D)$. Also

$$d(\{\nu|g_1 \xrightarrow{\nu} E\}\circ\mathcal{D}_D[\![s]\!], \{\nu|g_1 \xrightarrow{\nu} E\}\circ\tilde{\Psi}_D(s)) \leqslant \text{(since } \epsilon\notin\{\nu|g_1 \xrightarrow{\nu} E\})$$

$$\tfrac{1}{2}\, d(\mathcal{D}_D[\![s]\!], \tilde{\Psi}_D(s)) \leqslant \tfrac{1}{2}\, d(\mathcal{D}_D, \tilde{\Psi}_D)$$

Putting these two inequalities together, we have shown that

$$d(\mathcal{D}_D[\![g_1;s]\!], \tilde{\Psi}_D(g_1;s)) \leqslant \tfrac{1}{2}\, d(\mathcal{D}_D, \tilde{\Psi}_D)$$

(We use here that, for d any Hausdorff metric, $d(X_1\cup X_2, Y_1\cup Y_2)\leqslant\max\{d(X_i, Y_i)|i=1,2\}$.)

STAGE 2. We now show that $d(\mathcal{D}_D[\![s]\!], \tilde{\Psi}_D(s))\leqslant\frac{1}{2}\, d(\mathcal{D}_D, \tilde{\Psi}_D)$, for *all* s, by induction on the complexity of s. For each case this proceeds as in stage 1, except for the case $s\equiv x$. Then $d(\mathcal{D}_D[\![x]\!], \tilde{\Psi}_D(x))=d(\mathcal{D}_D[\![g]\!], \tilde{\Psi}_D(g))\leqslant$ (since g is guarded, stage 1 applies) $\frac{1}{2}\, d(\mathcal{D}_D, \tilde{\Psi}_D)$. Altogether, we have $d(\mathcal{D}_D[\![s]\!], \tilde{\Psi}_D(s))\leqslant\frac{1}{2}\, d(\mathcal{D}_D, \tilde{\Psi}_D)$, for all s. Hence, $d(\mathcal{D}_D, \tilde{\Psi}_D)=0$, as was to be shown. □

We have one lemma still to be filled in. For convenience, we use the notation (for $p,q\in Q$) $p\subseteq q$ as short hand for $p+q=q$.

LEMMA 9.4. *For each* $s\in L_6$,

$$\{\nu|s \xrightarrow{\nu} E\} \subseteq \mathcal{D}_D[\![s]\!]$$

PROOF. We introduce the auxiliary relation $s_1 \twoheadrightarrow_D s_2$ by the transition system

$$x \twoheadrightarrow_D g, \quad \text{for} \quad x \Leftarrow g \quad \text{in} \quad D$$

$$s_1+s_2 \twoheadrightarrow_D s_1$$

$$s_1+s_2 \twoheadrightarrow_D s_2$$

$$\frac{s_1 \twoheadrightarrow_D s_2}{C[s_1] \twoheadrightarrow_D C[s_2]}, \quad \text{for each arbitrary } L_6\text{-context } C[\cdot]$$

It is direct from the definition of $\twoheadrightarrow_D$ that, if $s_1 \twoheadrightarrow_D s_2$ holds, then $\mathcal{D}_D[\![s_1]\!] \supseteq \mathcal{D}_D[\![s_2]\!]$. We use $\twoheadrightarrow_D$ in the formulation of the following straightforward

CLAIM. If $s \xrightarrow{v} E$ then either
- $(v=a) \wedge (s \twoheadrightarrow_D a)$, or
- there exist $s_1, s_2, v_1, v_2 \in A^+$ such that $v = v_1.v_2$, $s_1 \xrightarrow{v_1} E$, $s_2 \xrightarrow{v_2} E$, and $s \twoheadrightarrow_D s_1;s_2$

We now prove the assertion of the lemma by induction on the length of v. If $v=a$, we have $a \in \mathcal{D}_D[\![a]\!] \subseteq \mathcal{D}_D[\![s]\!]$. If $v=v_1v_2$ $(v_1, v_2 \in A^+)$, then $s_1 \xrightarrow{v_1} E$, $s_2 \xrightarrow{v_2} E$, and $s \twoheadrightarrow_D s_1;s_2$. By induction, $v_1 \in \mathcal{D}_D[\![s_1]\!]$, $v_2 \in \mathcal{D}_D[\![s_2]\!]$, and we obtain $v_1v_2 \in \mathcal{D}_D[\![s_1;s_2]\!]$. Since $\mathcal{D}_D[\![s_1;s_2]\!] \subseteq \mathcal{D}_D[\![s]\!]$, the desired result follows. □

Altogether, we have completed the investigation of the transition system T_6, establishing that increasing the grain size in its transitions does not affect the associated operational semantics for $\mathcal{O}_6$.

ACKNOWLEDGEMENTS

Our paper owes much to the insights and criticisms of the Amsterdam Concurrency Group, consisting of Frank de Boer, Arie de Bruin, Joost Kok, John Meyer, Jan Rutten and Erik de Vink. We have already acknowledged above specific contributions from Arie de Bruin, Joost Kok and Erik de Vink. Joost Kok showed that a previous version of the transition system for L_6 did not work, and Jan Rutten showed how to correct it. We are grateful to Erik de Vink for detailed criticism on a draft of our paper.

APPENDIX

We provide a brief sketch of the translation of a rudimentary (and/or) parallel logic program into, for example, the language L_4. The approach followed in the translation is a (considerably) simplified version of the translation (due to Joost Kok) as described in [BK].

Let $a, a_1, \cdots, \bar{a}, \cdots$ be elements of $\mathcal{Atom}$, the set of (logical) atoms as used in logic programming. We consider *clauses* c of the form $a \leftarrow a_1 \wedge \cdots \wedge a_n$ $(n \geqslant 0)$, *programs* π which consist of a finite set of clauses $\{c_1, \cdots, c_k\}$, $k \geqslant 1$, and *goals* of the form $\bar{a}_1, \wedge \cdots \wedge \bar{a}_m$, $m \geqslant 0$. We provide a translation into L_4 of a pair $<\pi, g>$. The translation assumes a version of L_4 (with corresponding semantics) which works for arbitrary interpretations (rather than for no interpretation). That is, we assume a set Σ of states, and interpret the elementary actions in A as, possibly partial, state transformations. One further technical step is required to cope with possible clashes between (individual) variables in the clauses or goal. We assume that the set of individual variables $\mathcal{Var}$ is partitioned into disjoint sets $\mathcal{Var}_\alpha$, with $\alpha \in \mathbb{N}^*$, the set of all finite, possibly empty, sequences of natural numbers. Moreover, we assume that all individual variables in π and g are initially from $\mathcal{Var}_\epsilon$, and we assume injections $\alpha: \mathcal{Var}_{\bar{\alpha}} \to \mathcal{Var}_{\bar{\alpha}.\alpha}$, for each $\alpha, \bar{\alpha} \in \mathbb{N}^*$. The injections α are extended in the natural way to the atoms in $\mathcal{Atom}$. We now describe the translation:

- For A we take $\mathcal{Atom} \times \mathcal{Atom}$
- For $\mathcal{Prvar}$ we take $\mathcal{Atom} \times \mathbb{N}^*$
- For Σ we take the set of substitutions (in the usual sense of logic programming)
- As interpretation of an elementary action (a_1, a_2) we take $[\![(a_1, a_2)]\!](\sigma) = mgu(a_1, \sigma(a_2)) \circ \sigma$, where *mgu* denotes a fixed most general unifier
- We define the auxiliary mapping $trl: \mathcal{Clause} \times \mathcal{Prvar} \to L_4^g$ by putting

$$trl(a \leftarrow a_1 \wedge \cdots \wedge a_n, (\bar{a}, \alpha)) = (\alpha(a), \bar{a}); ((\alpha(a_1), \alpha.1) \| \cdots \| (\alpha(a_n), \alpha.n))$$

- Let $\pi=\{c_1, \cdots, c_k\}$. Take for the set of L_4-declarations D:

  $$D \equiv <(\bar{a}, \alpha) \Leftarrow trl(c_1,(\bar{a}, \alpha))+ \cdots +trl(c_k,(\bar{a}, \alpha))>_{(\bar{a}, \alpha)\in \mathcal{P}var}$$

  Note that, returning for a moment to the general L_4 syntax, D is of the form

  $$<x \Leftarrow a_1;(x_{11}\| \cdots \|x_{1n_1})+ \cdots +a_k;(x_{k1}\| \cdots \|x_{kn_k}>_{x\in \mathcal{P}var}$$

- Finally, take as translation of $<\pi, g>$ the L_4-program

  $$<D|(\bar{a}_1, 1)\| \cdots \|(\bar{a}_m,m)>.$$

REFERENCES

[ABKR] P. AMERICA, J.W. DE BAKKER, J.N. KOK, J.J.M.M. RUTTEN. *A denotational semantics of a parallel object-oriented language,* CWI Report CS-R8626, to appear in Information and Computation.

[AR] P. AMERICA, J.J.M.M. RUTTEN, *Solving reflexive domain equations in a category of complete metric spaces,* in Proc. of the Third Workshop on Mathematical Foundations of Programming Language Semantics, (M. Main, A. Melton, M. Mislove, D. Schmidt, eds.) LNCS 298, Springer, pp. 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.

[AP] K. APT, G. PLOTKIN, *Countable nondeterminism and random assignment,* Journal of the Association for Computing Machinery, Vol. 33, No. 4, (1986) 724-767.

[ABe] B. ARBAB, D.M. BERRY, *Operational and denotational semantics of PROLOG,* Journal of Logic Programming 4 (1987) 309-330.

[BBKM] J.W. DE BAKKER, J.A. BERGSTRA J.W. KLOP, J.-J.CH. MEYER, *Linear time and branching time semantics for recursion with merge,* TCS 34 (1984) 135-156.

[BK] J.W. DE BAKKER, J.N. KOK, *Uniform Abstraction, Atomicity and Contractions in the Comparative Semantics of Concurrent Prolog,* Report CS-R8834, Centre for Mathematics and Computer Science, Amsterdam (1988), also to appear in Proc. Fifth Generation Computer Systems, Tokyo, 1988.

[BKMOZ] J.W. DE BAKKER, J.N. KOK, J.-J.CH. MEYER, E.-R. OLDEROG, J.I. ZUCKER, *Contrasting themes in the semantics of imperative concurrency,* in Current Trends in Concurrency: Overviews and Tutorials (J.W. de Bakker, W.P. de Roever, G. Rozenberg, eds.), Lecture Notes in Computer Science, Vol. 224, Springer (1986) 51-121.

[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.

[BMO] J.W. DE BAKKER, J.-J.CH. MEYER, E.-R. OLDEROG, J.I. ZUCKER, *Transition systems, metric spaces and ready sets in the semantics of uniform concurrency,* Journal of Comp. Syst. Sc. (1988), 158-224.

[BZ] J.W. DE BAKKER, J.I. ZUCKER, *Processes and the denotational semantics of concurrency,* Inform. and Control 54 (1982) 70-120.

[Be] L. BECKMANN, *Towards a formal semantics for concurrent logic programming languages,* Proc. Third Int. Conference on Logic Programming (E. Shapiro, ed.), LNCS 225, Springer, (1986), 335-349.

[BeK1] J.A. BERGSTRA, J.W. KLOP, *A convergence theorem in process algebra,* Report CS-R8733, Centre for Mathematics and Computer Science (1987).

[BeK2] J.A. BERGSTRA, J.W. KLOP, *Bisimulation semantics,* 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.), LNCS, Springer, to appear.

[BoKPR] 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.

[BHR] S.D. BROOKES, C.A.R. HOARE, A.W. ROSCOE, *A theory of communicating sequential processes,* J. ACM 31 (1984) 499-560.

[Br] A. DE BRUIN, *Exercises in continuation semantics: jumps, backtracking, dynamic networks,* Dissertation, Free University, 1986.

[BrV] A. DE BRUIN, E.P. DE VINK, *Continuation semantics for PROLOG with cut,* Technical Report, Department of Computer Science, Free University of Amsterdam, to appear.

[DM] S.K. DEBRAY, P. MISHRA, *Denotational and operational semantics for PROLOG,* Journal of Logic Programming 5 (1988), 61-91.

[Du] J. DUGUNDJI, *Topology,* Allen and Bacon, Rockleigh, N.J. 1966.

[En] R. ENGELKING, *General topology,* Pòlish Scientific Publishers 1977.

[FL] M. FALASCHI, G. LEVI, *Operational and fixpoint semantics of a class of committed-choice languages,* Technical Report, Dipartimento di Informatica, Univ. di Pisa, 1988.

[FLMP] M. FALASCHI, G. LEVI, M. MARTELLI, C. PALAMIDESSI, *Declarative modeling of the operational behaviour of logic languages,* Technical Report, Dipartimento di Informatica, Univ. di Pisa, 1987.

[FOM] K. FURUKAWA, A. OKUMURA, M. MURAKAMI, *Unfolding rules for GHC programs,* in D. Bjorner, A.P. Ershov and N.D. Jones, eds., Workshop on Partial Evaluation and Mixed Computation, Gl. Avernaes, Denmark (1987), to appear in New Generation Computing.

[GCLS] R. GERTH, M. CODISH, Y. LICHTENSTEIN, E. SHAPIRO, *Fully abstract denotational semantics for Concurrent Prolog,* Proc. 3rd Symp. on Logic in Computer Science, Edinburgh (1988), 320-335.

[Gi] G. GIERZ, K.H. HOFMANN, K. KEIMEL, J.D. LAWSON, M. MISLOVE, D.S. SCOTT *A compendium of continuous lattices,* Springer-Verlag, 1980.

[Ha] H. HAHN, *Reelle Funktionen,* Chelsea, New York, 1948.

[HP] M. HENNESSY, 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).

[Ke] R.M. KELLER, Formal verification of parallel programs, Comm. ACM 19 (1976) 371-384.

[Ko] J.N. KOK, *A compositional semantics for Concurrent Prolog,* Proc. STACS 1988, LNCS 294, Springer, pp. 373-388.

[KR] J.N. KOK, J.J.M.M. RUTTEN, *Contractions in comparing concurrency semantics,* Proc 15th ICALP (T. Lepistö, A. Salomaa, eds.) LNCS 317, Springer (1988) 317-332.

[Kw] R.A. KOWALSKI, Algorithm = Logic + Control, Comm. ACM 22 (1979), pp. 424-435.

[Le] G. LEVI, *A new declarative semantics of Flat Guarded Horn Clauses,* Technical 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.

[L1] J.W. LLOYD, *Foundations of Logic Programming,* Springer (1984), (Second edition 1987).

[P11] G.D. PLOTKIN, *A powerdomain construction,* SIAM Journal of Computing, Vol. 5, no. 3, (1976) 452-487.

[P12] G.D. PLOTKIN, *A structural approach to operational semantics,* Report DAIMI FN-19, Comp. Sci. Dept., Aarhus Univ. 1981.

[P13] G.D. PLOTKIN, *An operational semantics for CSP,* in : D. Bjørner (ed.): Formal

description of programming concepts II, North-Holland (1983) 199-223.

[Ri] G.A. RINGWOOD, *Parlog 86 and the dining logicians,* Comm. ACM, Vol. 31 (1988) 10-25.

[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,* Techn. 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).

[Vi] E.P. DE VINK, *Equivalence of an operational and denotational semantics for a Prolog-like language with cut,* Technical Report IR 151, Free University, Amsterdam, 1988.

[VV] E. DE VINK, S. VAN VEEN, *Semantics of logic programming,* Report CS-N8508, Centre for Mathematics and Computer Science, Amsterdam (1985).