Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

S. van Veen, E. de Vink

Semantics of logic programming

Department of Computer Science Note CS-N8508 September

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.).

Semantics of Logic Programming

Simon van Veen
Free University, Amsterdam

Erik de Vink
University of Amsterdam

In this note several semantics of logic programming are given and equivalence of these semantics is proven. The technique of transition systems is used to describe the semantics of a subset of Prolog and Concurrent Prolog. A notion of fairness is introduced in order to model infinite computations of logic programs.

PROLOG

This paper is the result of our joint stay at the Centre for Mathematics and Computer Science (CWI). For a year we have worked on the project "Logic Programming" together with Jaco de Bakker, John-Jules Meyer, Joost Kok and Jan Rutten. In this group we discussed several papers and ideas concerning logic programming. The project has led to this paper.

First we will give a brief description of what we have done in the various chapters. In chapter 1 we give a general introduction to logic programming. Something is told about the way one can interpret logic programming. Chapter 2 describes several semantics of logic programming and equivalence of these semantics is proven. Chapter 3 describes semantics for a variant of logic programming, for a subset of Prolog and for a subset of Concurrent Prolog. A proof of equivalence of the semantics with respect to the subset of Prolog is given. Chapter 4 deals with infinite computations with logic programs. In chapter 5 some comments are given and some remarks about future research are made.

At the end of this prolog, we want to thank the members of the above mentioned group for their support and constructive criticism. Finally we thank "mcvax!boring" for processing this paper and making it into a readable form.

Note CS-N8508
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Chapter 1

Introduction to Logic Programming

Logic programming began in the early 1970's as a direct outgrowth of earlier work in automatic theorem proving and artificial intelligence. In 1965 Robinson [42] introduced the *resolution inference rule,* which is particularly well-suited to automation on a computer. Kowalski [27] showed us a procedural interpretation of a subset of the first order logic, which makes it very effective as a programming language. One of the most important practical outcomes of the research so far has been the language Prolog,[43]. Prolog is based on the so called *Horn clause* subset of logic.

One of the main ideas of logic programming is that an algorithm consists of two disjoint components, namely the *logic* and the *control.* The logic is the statement of *what* the problem is and the control is the statement *how* to solve it. The ideal of logic programming is that a programmer should only have to specify the logic. The control should be done by the logic programming system.
Unfortunately this ideal has not yet been achieved with current logic programming systems. In order to achieve the ideal of logic programming we have to overcome two problems. The *control problem:* For example in the logic programming language Prolog, a programmer provides a lot of control information by the ordering of the clauses and atoms in the clauses and by extra-logical control features, such as cut. The *negation problem:* With the Horn clause subset of logic it is possible to prove the validity of a positive literals, but nothing can be said about negative ones.

In this chapter we give an informal description of the procedural interpretation of logic programming. Besides the procedural interpretation there are two other interpretations of logic programming, viz. the *database-interpretation,* (which we will not deal with) and the *process-interpretation.* In the process-interpretation a goal is regarded as a system of concurrent processes. A step in the computation is the reduction of a process to a system of processes. Shared variables act as communication channels between processes. One such interpretation is Concurrent Prolog of Shapiro, which we will discuss in chapter 3.

Some definitions

The language is a subset of first order predicate calculus. In our definition we use a special kind of wellformed formulas, viz. the Horn clause subset. As usual we assume formulas to be built up of the well known connectives $\neg$ and $\vee$, and the quantifier $\forall$.

Let A be an alphabet of a first order language. Let VAR be a denumerable collection of variables. Let $TERM$ (with typical elements s, t, ..) resp. $ATOM$ (with typical elements a,..) denote the collection of terms resp. the collection of all atoms over the alphabet A and with variables in VAR.

A literal is an atom or the negation of an atom. A clause is a disjunction of literals. Variables are implicitly universally quantified.

A *substitution* is a map $\theta : VAR \rightarrow TERM$. If $a(x_1,..,x_n)$ is an atom with variables $x_1,..,x_n$, then $\theta(a(x_1,..,x_n)) = a(\theta(x_1),..,\theta(x_n))$

A *unifier* of two atoms a_1 and a_2 is a substitution θ, such that $\theta(a_1) = \theta(a_2)$. A *most general unifier* θ is a unifier of two atoms a_1 and a_2, such that for every unifier ϕ there is a substitution ψ, such that $\psi \circ \theta = \phi$. If a unifier exists then also the most general unifier exists.

Lloyd gives a nice and clear unification algorithm in [35]. The unification algorithm is a partial function *mgu*: $ATOM \times ATOM \rightarrow (VAR \rightarrow TERM)$, such that $mgu(a_1, a_2)$ is a most general unifier of the atoms a_1 and a_2, if it exists and otherwise it is undefined. We assume without loss of generality that if $\theta = mgu(a_1,a_2)$, then $\theta(x) = x$ for all $x \notin var(a_1,a_2)$.

HORN CLAUSE LOGIC

In the next sections we only deal with a specific kind of clauses, namely the *Horn clauses.* These are defined as follows:

- A definite clause is a clause which contains only one positive atom: $a_0 \vee \neg a_1 \vee .. \vee \neg a_n$. Another notation is: $a_0 \leftarrow a_1 \wedge .. \wedge a_n$. If n=0, then the clause is called an axiom. If n>0, then the clause is called a rule.
- A negative clause or goal clause is a clause, which contains only negative atoms: $\neg a_1 \vee .. \vee \neg a_n$. Another notation is: $\leftarrow a_1 \wedge .. \wedge a_n$. If n=0, then the clause is called the empty goal. We denote the empty goal by **true**.

In the definite clause $a_0 \leftarrow a_1 \wedge .. \wedge a_n$ we call a_0 the head of the clause and $a_1 \wedge .. \wedge a_n$ the body.

A *logic program* is a set of definite clauses.

The *resolution inference rule* due to Robinson [42] is in essence a rewrite rule, defined as follows: given a goal clause $\leftarrow a_1 \wedge .. \wedge a_n$. Let a_i be the selected atom to be rewritten. Let $a \leftarrow b_1 \wedge .. \wedge b_m$ be a definite clause, with no variables with $\leftarrow a_1 \wedge .. \wedge a_n$ in common. such that the most general unifier of a_i and a exists, say θ. Then the clause $\leftarrow \theta(a_1 \wedge .. \wedge a_{i-1} \wedge b_1 \wedge \ \wedge b_m \wedge a_{i+1} \wedge .. \wedge a_n)$ is obtained by resolution of a_i against $a \leftarrow b_1 \wedge .. \wedge b_m$. Given a logic program P and a goal clause $\leftarrow a_1 \wedge .. \wedge a_n$, we try to find by successive applications of the resolution inference rule an instance of $a_1 \wedge .. \wedge a_n$, which is a consequence of P. Then we derive that $\leftarrow a_1 \wedge .. \wedge a_n$ and P are inconsistent. Hence $\leftarrow a_1 \wedge .. \wedge a_n$ is refutable with respect to P.

A *derivation* of a goal clause G_0 with respect to a logic program P, i.e. a set of definite clauses $\{ \alpha_i \mid 1 \leqslant i \leqslant q \}$ is a sequence of negative clauses $G_0, G_1,.., G_k$, such that G_i is obtained by resolution, of a conjunct of G_{i-1} against some α_j with mgu θ_i. A derivation has an associated substitution, viz. the composition $\theta_k \circ .. \circ \theta_1$ of the most general unifiers determined within each step of the derivation.

A derivation can be infinite or finite. When it is finite it can be successful or failing. A finite derivation is successful when the last goal in the derivation yields the empty clause. In this case the associated substitution is called the computed answer substitution. The derivation is called a refutation of the initial goal clause. It fails when the selected atom in the last goal clause can not be rewritten by any of the clauses α_i.

In the rewrite process the selected atom will be specified by a so called *computation rule* (for example one can always choose the leftmost atom in the goal). With respect to successful computations computation rules are equivalent, (i.e. they give equivalent answer substitutions). Although a given computation rule defines the selected atom in a goal, there are several derivations possible since that atom might be unifiable with the heads of different clauses.

A *search tree* is the representation of all the derivations which are possible for a given goal clause and a given computation rule. The tree has as nodes goal clauses. The descendents of a node are the goal clause of this node, in which the selected atom has been replaced by the body of a clause whose head unifies the selected atom. The most general unifier is applied to the new (goal) node. This is

done for all the definite clauses, whose heads unifies the selected atom. It follows that different computation rules generally lead to different search trees.

A *search rule* is a strategy to find the finite path in a search tree, which correspond with refutations of the initial goal.

For example the logic programming language Prolog uses as computation rule: leftmost and as search rule: depth first (with ordering on the definite clauses). In chapter 3 of this paper we will give the operational and denotational semantics of a subset of Prolog with these computation and search rule.

THE PROCEDURAL INTERPRETATION OF LOGIC PROGRAMS

In the procedural interpretation we regard the set of definite clauses as procedure declarations. The head of the clause is the name of the procedure with "structured" formal parameters. The body of the clause is a set of procedure calls. The goal clause is the main program, which is a set of procedure calls. Finally the empty clause is the halt statement.

The inference rule (the resolution strategy) is similar to the dynamic copy rule for procedures, i.e. the replacement of the procedure call by a copy of its body. Parameter passing and value return are done by unification of a procedure call with a procedure head. A given variable in a procedure head can behave differently with different procedure calls.

A variable can act partially as input and partially as output parameter: a partial value can be passed to a procedure which then computes a further approximation which is passed back to the caller. There are also local variables, which only occur in the procedure body and not in the procedure head.

The described procedural interpretation of logic programming is thus similar to the operational interpretation of the imperative programming languages.

Chapter 2

The Semantics of Logic Programming

INTRODUCTION

In this chapter various semantics of logic programming are described: a model-theoretic semantics, a fixpoint semantics, an operational semantics, a leftmost semantics and a denotational semantics. Here the domains of interpretation are sets of ground atoms and sets of answer subtitutions. Also an operational and denotational semantics is given with multisets of answer substitution as domain.

2.1. Various semantics of logic programming

DEFINITIONS

Let VAR be a denumerable collection of variables, with typical element x. Let A be an alphabet of a first order language. Let $TERM$, resp. $ATOM$ denote the collection of all terms, resp. the collection of all atoms over the alphabet A and with variables in VAR. Let $CONJ$ denote the collection of all conjunctions of atoms. Especially the empty conjunction, denoted by **true** is a member of $CONJ$. t, a, resp. c are typical elements of $TERM$, $ATOM$, resp. $CONJ$.

Let $\{ VAR_i \mid 0 \leqslant i \}$ be a partition of VAR in infinite, pairwise disjoint subsets. Define $VAR_{\leqslant m} = \cup_{i=0}^{m} VAR_i$. The collection $CLAUSE$ of all definite Horn clauses with typical element α is defined by $\alpha ::= a \leftarrow c$ where $var(a,c) \subseteq VAR_0$. The collection $PROG$ of all logic programs with typical element P is defined by $P ::= \alpha \mid (P' \cup P'')$. The collection $SENT$ of all logic programs with input, with typical element S is defined by $S ::= P \leftarrow c$. A renaming is a permutation of VAR. Fix for each collection VAR_i some renaming ψ_i from VAR_0 onto VAR_i. If P is a logic program, then P_i is the logic program obtained by replacement of each occurrence of $x \in VAR_0$ by $\psi_i(x) \in VAR_i$. If $\{X', X''\} \subseteq CONJ$ or $CLAUSE$ then X' is called a variant of X'' if there exists a renaming ψ of VAR such that X' can be obtained from X'' by replacement of each occurrence of a variable x by $\psi(x)$.

Every $f : VAR \rightarrow TERM$ induces a mapping $f : ATOM \rightarrow ATOM$ in a straightforward manner. If $f : VAR \rightarrow TERM$ then $dom(f) = \{ x \in VAR \mid f(x) \neq x \}$ and $rge(f) = \cup \{ var(f(x)) \mid x \in dom(f) \}$. The class $SUBST$ with typical element ϕ is defined by $SUBST = \{ f : VAR \rightarrow TERM \mid dom(f) \cap rge(f) = \varnothing \}$. A variable-free term, atom or conjunction is called a ground term, atom or conjunction. If $a \in ATOM$ then the ground closure $[a]$ of a is defined by $[a] = \{ \phi(a) \in ATOM \mid \phi \in SUBST \text{ s.t. } var(\phi(a)) = \varnothing \}$. Let NUM denote the set of natural numbers.

MODEL-THEORETIC SEMANTICS

In this section we stress the logical component of logic programming. When we regard logic programs just as first order theories, we can use the standard semantics of predicate logic to give meaning to them. However, we are not interested in arbitrary models of logic programs. In order to make a comparison between the model-theoretic semantics and other ones we construct models of logic programs with a special universe, where we have control over the elements.

Let A be an alphabet of a first order language. The Herbrand universe HU of A is the collection of all ground terms of A. The Herbrand base HB of A is the collection of all ground atoms of A. A Herbrand interpretation is just a subset of HB. Given a logic program, the Herbrand universe, the Herbrand base and the Herbrand interpretations are taken with respect to the alphabet of P.
Let C, F resp R be the collection of constants, function symbols and relation symbols of A. A Herbrand interpretation I induces an interpreting structure of A, $(HU,*)$, where
1. if $c \in C$ then $c^* = c$;
2. if $f \in F$ k-ary, then $f^*(t_1,..,t_k) = f(t_1,..,t_k)$;
3. if $r \in R$ k-ary, then $r^* = \{ (t_1,..,t_k) \mid r(t_1,..,t_k) \in I \}$.
Satisfaction in such a structure is as usual. If P is a logic program with alphabet A, then a Herbrand interpretation which satisfies the logic program P itself is called a Herbrand model of P.

Apparently, every logic program has a Herbrand model, e.g. HB itself, because of the special form of definite Horn-clauses. Moreover, Herbrand models have the intersection property, i.e. the intersection of all Herbrand models is again a Herbrand model. Hence we can talk about the least Herbrand model of P.

(2.1) DEFINITION Let P be a logic program. Then the model-theoretic semantics $\mathfrak{M}(P)$ of P is defined by $\mathfrak{M}(P) = \cap\{ M \mid M \text{ is a Herbrand model of P} \}$.

FIXPOINT SEMANTICS

If we look at a Herbrand model of a logic program P, then the axioms and rules of P are satisfied by the induced structures. So intuitively the several relations of the structures contain the ground instances of the axioms and are closed with respect to ground instances of the rules. We formalize this intuition by the definition of a transformation of the powerset of the Herbrand base, that reflects the underlying idea of closedness with respect to a logic program.

(2.2) DEFINITION Let P be a logic program. We associate with P a mapping $\mathfrak{T}:\mathfrak{P}(HB) \to \mathfrak{P}(HB)$, $\mathfrak{T}(I) = \{ \phi(a_0) \in HB \mid a_0 \leftarrow a_1 \wedge .. \wedge a_r \in P, r \geq 0; \phi \in SUBST; \phi(a_1),..,\phi(a_r) \in I \}$.

Because of the simple structure of the complete lattice $<\mathfrak{P}(HB), \subseteq>$, it is easy to see that $\mathfrak{T}$ is a continuous map of $\mathfrak{P}(HB)$ to itself. Especially we have both $lfp(\mathfrak{T}) = \cap\{I \in \mathfrak{P}(HB) \mid \mathfrak{T}(I) \subseteq I \}$ and $lfp(\mathfrak{T}) = \cup_{n=0}^{\infty} \mathfrak{T}^n(\varnothing)$. Another consequence of the definition of $\mathfrak{T}$ is the equivalence for each Herbrand interpretation I of "I satisfies P" on the one hand, and "$\mathfrak{T}(I) \subseteq I$" on the other.
Now we are ready to define the so-called fixpoint semantics of a logic program. It will play an intermediate role between the model-theoretic and operational approach.

(2.3) DEFINITION Let P be a logic program and let $\mathfrak{T}$ be the transformation of $\mathfrak{P}(HB)$ associated with P. The fixpoint semantics $\mathfrak{F}(P)$ of P is defined by $\mathfrak{F}(P) = lfp(\mathfrak{T})$.

The next theorem justifies the introduction of the transformation $\mathfrak{T}$. It states that the fixpoint semantics and the model-theoretic semantics coincide.

(2.4) THEOREM Let P be a logic program. Then $\mathfrak{M}(P) = \mathfrak{F}(P)$.

PROOF Let a be ground atom. Then $a \in \mathfrak{M}(P)$ iff $a \in \cap\{ M \subseteq HB \mid M \text{ is a Herbrand model of P} \}$ iff $a \in \cap\{ I \subseteq HB \mid \mathfrak{T}(I) \subseteq I \}$ iff $a \in lfp(\mathfrak{T})$ iff $a \in \mathfrak{F}(P)$. □

OPERATIONAL SEMANTICS

In this section we focus on the computational component of logic programming. We consider a logic program as a set of rewrite rules with side effects. A computation by a logic program is just a sequence of applications of the rewrite rules on a goal with the side effects globally registrated.
The rewrite procedure starting with some initial goal eventually terminates. In this case we are interested in the result of the computation. Sometimes the rewrite procedure diverges, i.e. there is an infinite sequence of applications of the rules. We want an operational semantics that reflects both possibilities of computational behaviour. (Hence we neglect the situations where the rewrite procedure deadlocks.)

First we formalize the notion of a rewrite procedure associated with a logic program. To this end we define the index of a formula Φ, written as $index(\Phi)$, as the least non-negative integer m such that $var(\Phi) \subseteq VAR_{\leqslant m}$, and the index of a substitution ϕ, also written as $index(\phi)$, as the least non-negative integer m such that $dom(\phi)$, $rge(\phi) \subseteq VAR_{\leqslant m}$. (Note that sometimes index(ϕ) is not defined.) We define the set of goals $GOAL$ with respect to some fixed alphabet by $GOAL = \{ <c,\phi,m> \in CONJ \times SUBST \times NUM \mid index(c), index(\phi) \leqslant m \}$. Given a program P the set $GOAL$ is taken with respect to the alphabet of P. Typical elements of $GOAL$ are denoted as g. We consider $CONJ$ as a subset of $GOAL$ via the embedding $c \mapsto <c,id,index(c)>$.

(2.5) DEFINITION Let P be a logic program. The binary relation ${}_P\!\rightarrow$ on $GOAL$ induced by P is defined by:
$<\wedge_{i=1}^{s} a_i,\phi,m> \; {}_P\!\rightarrow \; <\wedge_{i=1}^{j-1} a_i \wedge \wedge_{i=1}^{r} a_i' \wedge \wedge_{i=j+1}^{s} a_i, \theta \circ \phi, m+1>$ iff for some j, $1 \leqslant j \leqslant s$ and some clause $a_0' \leftarrow a_1' \wedge .. \wedge a_r' \in P_{m+1}$, $(r \geqslant 0)$, $\theta = mgu(\phi(a_j), a_0')$ exists.

We say that $<\wedge_{i=1}^{s} a_i,\phi,m>$ produces $<\wedge_{i=1}^{j-1} a_i \wedge \wedge_{i=1}^{r} a_i' \wedge \wedge_{i=j+1}^{s} a_i, \theta \circ \phi, m+1>$ via θ and $a_0' \leftarrow a_1' \wedge .. \wedge a_r'$. Because we select just one mgu for two unifiable atoms via the partial function mgu, we have for each $g \in GOAL$: $\{ g' \mid g \; {}_P\!\rightarrow g' \}$ is finite.

${}_P\!\rightarrow^*$ denotes the reflexive and transitive closure of ${}_P\!\rightarrow$ in $GOAL$. We say $g \; {}_P\!\rightarrow^* \Omega$ with Ω a special symbol not in $GOAL$, if there exists an infinite sequence $(g_i)_{i=0}^{\infty}$ such that $g_0 = g$ and g_{i-1} produces g_i for $i \geqslant 1$. We say $g \; {}_P\!\rightarrow^* g'$ via ϕ if there exists a sequence $(g_i)_{i=0}^{n}$ in $GOAL$ such that $g_0 = g$, $g_n = g'$ and $\phi = \theta_n \circ .. \circ \theta_1$, where $g_{i-1} \; {}_P\!\rightarrow g_i$ via θ_i. Intuitively $g \; {}_P\!\rightarrow^* g'$ via ϕ, if there exists a sequence of applications of the rewrite rules that yields g' and has side effect ϕ. This ϕ is in fact a composition of mgu's and the number of involved mgu's equals the number of rewritings.
The third coordinate of elements of $GOAL$ serves as a counter. It assures us that no clashes of variables will happen. The variant of the clause being used introduces only variables which did not occur before.

A finite derivation Δ from $c \in CONJ$ of index m is a finite sequence $(g_i)_{i=0}^{n}$ such that: 1. $g_0 = <c,id,m>$; 2. $g_{i-1} \rightarrow g_i$, $1 \leqslant i \leqslant n$.
c is called the root of derivation Δ. A finite derivation is called successful, if it eventually yields an empty goal. The set $[\Delta(a)]$ computed by a finite derivation $\Delta = (g_i)_{i=0}^{k}$ from an atom a is defined as the ground closure $[\phi(a)]$ where ϕ as in $g_k = <c,\phi,n>$. We refer to ϕ as the result of the finite derivation.

A mapping $\rho: GOAL \rightarrow CONJ$ is called a computation-rule, if for each $g = <c,\phi,m> \in GOAL$, $\rho(g)$ is a conjunct of $\phi(c)$. A derivation $(g_i)_{i=0}^{n}$ is according to some computation-rule ρ if $g_{i-1} \rightarrow g_i$ by resolution of $\rho(g_{i-1})$.

(2.6) DEFINITION Let P be a logic program. The operational semantics $\mathcal{O}_P$ of P is defined by $\mathcal{O}_P$: $CONJ \rightarrow \mathcal{P}(SUBST \cup \{ \perp \})$ such that $\phi \in \mathcal{O}_P(c)$ iff for some $n \geqslant 0$ $c \; {}_P\!\rightarrow^* <\mathbf{true},\phi,n>$ and $\perp \in \mathcal{O}_P$ iff $c \; {}_P\!\rightarrow^* \Omega$.

So on the one hand a non-bottom element $\phi \in \mathcal{O}_P(c)$ represents the side effects caused by the successive rewritings in a terminating computation and on the other hand $\perp \in \mathcal{O}_P$ if there is an infinite computation starting with c.

The next proposition states that the subset of $SUBST \cup \{ \perp \}$ occurring as images of $\mathcal{O}_P$ are of a special form. If there are infinitely many terminating computations then there is also a diverging one.

(2.7) PROPOSITION Let P be a program. For each $c \in CONJ$: $\mathcal{O}_P(c)$ is finite or $\mathcal{O}_P(c)$ is infinite and contains $\bot$.

PROOF Trivial by König's lemma. □

PROPERTIES OF THE COMPUTATION MECHANISM.

In this section we establish some properties of the computation mechanism (with respect to some fixed logic program P). We will use these properties freely in the sequel. The first proposition enables us to glue computations together if we avoid clashes of variables. The second proposition states that a derivation step depends only on local information, i.e. on the variables of $\phi(a_i)$. The third proposition reflects the flexibility with respect to the renamings applied to the clauses in the program. The rectangle lemma states that if a goal has an instance which can be resolved successfully, then the goal itself can be resolved successfully. The fourth proposition states that in a derivation the order in which two atoms in a goal are resolved can be changed without effecting the result of the derivation essentially. The last proposition is a further elaboration of this phenomenon. The theorem of this section states that if a successful derivation from a goal exists according some computation-rule then every derivation from that goal according any computation-rule is successful.

(2.8) PROPOSITION Let $c_1, c_2 \in CONJ$ such that $var(\phi(c_1)) \cap var(\phi(c_2)) = \varnothing$. Suppose $<c_1,\phi,m> \rightarrow^* <d_1,\psi_1,m+n_1>$ and $<c_2,\phi,m> \rightarrow^* <d_2,\psi_2,m+n_2>$. Let ξ be the renaming which takes VAR_{m+i} onto VAR_{m+n_1+i}, and VAR_{m+n_1+i} onto VAR_{m+i}, $1 \leqslant i \leqslant n_1$. Define $\Psi = \xi \circ \psi_2 \circ \xi \circ \psi_1 \circ \phi$. Then $<c_1 \wedge c_2,\phi,m> \rightarrow^* <d_1 \wedge d_2,\Psi,m+n_1+n_2>$. □

(2.9) PROPOSITION Let $<\wedge_{i=1}^{r} a_i,\phi,m> \in GOAL$, $a_0' \leftarrow a_1' \wedge .. \wedge a_s' \in P_{m+1}$, $j \in \{1,..,r\}$. Then $<\wedge_{i=1}^{r} a_i,\phi,m> \rightarrow <\wedge_{i=1}^{j-1} a_i \wedge \wedge_{i=1}^{s} a_i' \wedge \wedge_{i=j+1}^{r} a_i, \theta \circ \phi, m+1>$ via mgu θ iff $<\wedge_{i=1}^{r} \phi(a_i),id,m> \rightarrow <\wedge_{i=1}^{j-1} \phi(a_i) \wedge \wedge_{i=1}^{s} a_i' \wedge \wedge_{i=j+1}^{r} \phi(a_i), \theta, m+1>$. □

(2.10) PROPOSITION Let $<\wedge_{i=1}^{r} a_i,\phi,m> \in GOAL$, $a_0' \leftarrow a_1' \wedge .. \wedge a_s' \in P_{m+1}$, $j \in \{1,..,r\}$, $k \geqslant 1$. Suppose $\theta = mgu(\phi(a_j),a_0')$ exists, so $<\wedge_{i=1}^{r} a_i,\phi,m> \rightarrow <\wedge_{i=1}^{j-1} a_i \wedge \wedge_{i=1}^{s} a_i' \wedge \wedge_{i=j+1}^{r} a_i, \theta \circ \phi, m+1>$ and let $\bar{a}_0 \leftarrow \bar{a}_1 \wedge .. \wedge \bar{a}_s \in P_{m+k+1}$ be a variant of $a_0' \leftarrow a_1' \wedge .. \wedge a_s'$. Then $\sigma = mgu(\phi(a_j),\bar{a}_0)$ exists and $<\wedge_{i=1}^{r} a_i,\phi,m+k> \rightarrow <\wedge_{i=1}^{j-1} a_i \wedge \wedge_{i=1}^{s} \bar{a}_i \wedge \wedge_{i=j+1}^{r} a_i, \sigma \circ \phi, m+k+1>$. □

(2.11) LEMMA (Rectangle lemma). Suppose $<\wedge_{i=1}^{r} a_i,\phi,\underline{m}> \rightarrow^* <c,\theta_l \circ .. \circ \theta_1 \circ \phi,m+l\underline{>}$ via mgu's $\theta_1,..,\theta_l$. Let $\phi' \in SUBST$ such that for some substitution ϕ of index $\leqslant$ m we have $\phi = \phi \circ \phi'$. Then $<\wedge_{i=1}^{r} a_i,\phi',m> \rightarrow^* <c, \theta_l' \circ .. \circ \theta_1' \circ \phi', m+l>$ via mgu's $\theta_1',..,\theta_l'$ and there exist $\psi \in SUBST$ such that $\theta_l \circ .. \circ \theta_1 \circ \phi = \psi \circ \theta_l' \circ .. \circ \theta_1' \circ \phi'$.

PROOF By induction on l. We only prove case $l+1 \leftarrow l$: Assume $<\wedge_{i=1}^{r} a_i,\phi,m> \rightarrow <\bar{c},\theta_1 \circ \phi,m+1>$ by resolution of a_j against $\bar{a}_0 \leftarrow \bar{a}_1 \wedge .. \wedge \bar{a}_t$ and $<\bar{c},\theta_1 \circ \phi,m+1\underline{>} \rightarrow^* <c, \theta_{l+1} \circ .. \circ \theta_1 \circ \phi, m+l+1>$. Since $\theta_1(\phi(\phi'(a_j)))\underline{\ } = \theta_1(\phi(a_j)) = \theta_1(\bar{a}_0) = \theta_1(\phi(\bar{a}_0))$, $\theta'_1 = mgu(\phi'(a_j),\bar{a}_0)$ exists and for some $\psi \circ \theta'_1 = \theta_1 \circ \phi$. Because $<\bar{c},\psi \circ \theta_1' \circ \phi',m+1> \rightarrow^* <c, \theta_{l+1} \circ .. \circ \theta_2 \circ \psi \circ \theta'_1 \circ \phi', m+l+1>$ we have by induction $<\bar{c},\theta_1' \circ \phi',m+1> \rightarrow^* <c, \theta'_{l+1} \circ .. \circ \theta'_1 \circ \phi', m+l+1>$ via mgu's $\theta'_2,..,\theta'_{l+1}$ and for some $\psi \in SUBST$ $\theta_{l+1} \circ .. \circ \theta_2 \circ \psi' \circ \theta_1' \circ \phi' = \psi \circ \theta'_{l+1} \circ .. \circ \theta'_1 \circ \phi'$. So $<\wedge_{i=1}^{r} a_i,\phi,m> \rightarrow^* <c, \theta_l' \circ .. \circ \theta_1' \circ \phi', m+l+1>$ via mgu's $\theta'_1,..,\theta'_{l+1}$ and there exists $\psi' \in SUBST$ such that $\theta_{l+1} \circ .. \circ \theta_1 \circ \phi = \psi \circ \theta'_{l+1} \circ$ □

(2.12) PROPOSITION (1). Let $a,a' \in ATOM$, ξ a renaming and $m \geqslant 0$ such that $a = \xi(a')$, $var(a,a') \subseteq VAR_{\leqslant m}$ and $index(\xi) \leqslant m$. Let $\bar{a} \in ATOM$ such that $var(\bar{a}) \subseteq VAR_{m+1}$. Suppose $\theta = mgu(a,\bar{a})$ exists. Then $\sigma = mgu(a',\bar{a})$ exists and for some renaming η we have $\theta \circ \xi = \eta \circ \sigma$ and $index(\eta) \leqslant m+1$.

(2). Let $<c,\phi,m>$, $<c',\phi',m> \in GOAL$ and ξ a renaming such that $\phi(c) = \xi(\phi'(c'))$ and $index(\xi) \leqslant m$. Suppose $<c,\phi,m> \rightarrow <\bar{c},\theta \circ \phi,m+1>$ by resolution against $\alpha \in P_{m+1}$ via mgu θ.

Then $<c',\phi',m> \rightarrow <\overline{c}',\sigma\circ\phi',m+1>$ by resolution against α via mgu σ and there exists a renaming η such that $\theta\circ\phi(\overline{c}) = \eta\circ\sigma\circ\phi'(\overline{c}')$, $\theta\circ\xi = \eta\circ\sigma$ and $index(\eta)\leqslant m+1$.

(3). Let $<c,\phi,m>$, $<c',\phi',m> \in GOAL$ and ξ be a renaming such that $\phi(c) = \xi(\phi'(c'))$ and $index(\xi)\leqslant m$. Suppose $(g_i)_{i=0}^{n}$ is a derivation from $<c,\phi,m>$ such that $g_{i-1}\rightarrow g_i$ by resolution against $\alpha^{(i)}$ via mgu θ_i, $(1\leqslant i\leqslant n)$. Then there exist a derivation $(g_i')_{i=0}^{n}$ from $<c',\phi',m>$ and a renaming η such that $g_{i-1}'\rightarrow g_i'$ by resolution against $\alpha^{(i)}$ via mgu σ_i, $(1\leqslant i\leqslant n)$, $\theta_n\circ..\circ\theta_1\circ\phi(c_n) = \eta\circ\sigma_n\circ ..\circ\sigma_1\circ\phi'(c_n')$, (with c_n,c_n' conjunctions in g_n,g_n'), $\theta_n\circ..\circ\theta_1\circ\xi = \eta\circ\sigma_n\circ..\circ\sigma_1$ and $index(\eta)\leqslant m+n$.

PROOF (1). Note $\theta\circ\xi(a') = \theta(a) = \theta(\overline{a}) = \theta\circ\xi(\overline{a})$. So σ exists and for some $\eta\in SUBST$: $\theta\circ\xi = \eta\circ\sigma$. Since $\sigma\circ\xi(a) = \sigma(a') = \sigma(\overline{a}) = \sigma\circ\xi(\overline{a})$ there exists $\zeta\in SUBST$ such that $\sigma\circ\xi = \zeta\circ\theta$. Without loss of generality $dom(\eta)\subseteq rge(\sigma)$, $dom(\zeta)\subseteq rge(\theta)$ and $index(\eta,\zeta)\leqslant m+1$. From $\sigma = \sigma\circ\xi\circ\xi = \zeta\circ\theta\circ\xi = \zeta\circ\theta\circ\sigma$ we conclude $\zeta\circ\eta|_{rge(\sigma)} = id$ and η is a renaming.

(2). Let a be a conjunct of c such that $\phi(a) = \xi\circ\phi'(a')$. Let $\overline{a}$ be the head of α. By part (1): $\sigma = mgu(\phi'(a'),\overline{a})$ exists and for some renaming η of index $\leqslant m+1$ we have $\theta\circ\xi = \eta\circ\sigma$. From $\theta\circ\phi(c) = \theta\circ\xi\circ\phi'(c') = \eta\circ\sigma\circ\phi'(c')$ and $\theta\circ\phi(\alpha) = \theta\circ\xi\circ\phi'(a) = \eta\circ\sigma\circ\phi'(a)$ we derive $\theta\circ\phi(\overline{c}) = \eta\circ\sigma\circ\phi'(\overline{c}')$.

(3). By induction on n. Case $n=0$: trivial. Case $n+1\leftarrow n$: Suppose $(g_i)_{i=0}^{n+1}$ is a derivation from $<c,\phi,m>$ such that $g_{i-1}\rightarrow g_i$ by resolution against $\alpha^{(i)}$ via mgu θ_i, $(1\leqslant i\leqslant n+1)$. By the induction hypothesis there exists a derivation $g_{i-1}'\rightarrow g_i'$ and a renaming ζ such that $g_{i-1}'\rightarrow g_i'$ by resolution against $\alpha^{(i)}$ via mgu σ_i, $(1\leqslant i\leqslant n)$, $\theta_n\circ..\circ\theta_1\circ\phi(c_n) = \zeta\circ\sigma_n\circ ..\circ\sigma_1\circ\phi'(c_n')$, $\theta_n\circ..\circ\theta_1\circ\xi = \zeta\circ\sigma_n\circ..\circ\sigma_1$ and $index(\zeta)\leqslant m+n$. By part 2.: $<c_n', \sigma_n\circ..\circ\sigma_1\circ\phi',m+n> \rightarrow <c_{n+1}', \sigma_{n+1}\circ..\circ\sigma_1\circ\phi',m+n+1>$ by resolution against $\alpha^{(n+1)}$ via mgu σ_{n+1} and there exists a renaming η such that $\theta_{n+}\circ..\circ\theta_1\circ\phi(c_n) = \zeta\circ\sigma_{n+1}\circ ..\circ\sigma_1\circ\phi'(c_n')$, $\theta_{n+1}\circ\zeta = \eta\circ\sigma_{n+1}$ and $index(\eta)\leqslant m+n+1$. Note $\theta_{n+1}\circ..\circ\theta_1\circ\xi = \theta_{n+1}\circ\zeta\circ\sigma_n\circ..\circ\sigma_1 = \eta\circ\sigma_{n+1}\circ..\circ\sigma_1$. We conclude that there exist a derivation $(g_i')_{i=o}^{n+1}$ from $<c',\phi',m>$ and a renaming η such that $g_{i-1}'\rightarrow g_i'$ by resolution against $\alpha^{(i)}$ via mgu σ_i, $(1i\leqslant n)$, $\theta_{n+1}\circ..\circ\theta_1\circ\phi(c) = \zeta\circ\sigma_{n+1}\circ\ \theta_{n+1}\circ..\circ\theta_1\circ\xi = \theta\circ\sigma_{n+1}\circ..\circ\sigma_1$ and $index(\eta)\leqslant m+n+1$. □

(2.13) PROPOSITION Let $g=<c,\phi,m>\in GOAL$ and a,a' conjuncts of c. Suppose $g\rightarrow g_1$ by resolution of $\phi(a)$ against $\alpha\in P_{m+1}$ via mgu θ_1 and $g_1\rightarrow g_2$ by resolution of $\theta_1\circ\phi(a')$ against $\alpha'\in P_{m+2}$ via mgu θ_2. Let $\xi: VAR_{m+1}\cup VAR_{m+2} \rightarrow VAR_{m+1}\cup VAR_{m+2}$ be canonical, $\overline{\alpha}=\xi(\alpha')\in P_{m+1}$, $\overline{\alpha}'=\xi(\alpha)\in P_{m+2}$. Then $g\rightarrow g_1'$ by resolution of $\phi(a')$ against $\overline{\alpha}$ via mgu σ_1 $g_1'\rightarrow g_2'$ by resolution of $\sigma_1\circ\phi(a)$ against $\overline{\alpha}'$ via mgu σ_2 and there exists a renaming η such that $\theta_2\circ\theta_1\circ\phi(c_2) = \eta\circ\sigma_2\circ\sigma_1\circ\phi(c_2')$, $index(\eta)\leqslant m+2$, and $\theta_2\circ\theta_1\circ\xi = \eta\circ\sigma_2\circ\sigma_1$.

PROOF Let $d,d',\overline{d},\overline{d}'$ be the head of $\alpha,\alpha',\overline{\alpha},\overline{\alpha}'$. We have $\theta_2\circ\theta_1\circ\xi\circ\phi(a') = \theta_2\circ\theta_1\circ\phi(a') = \theta_2(d') = \theta_2\circ\theta_1(d') = \theta_2\circ\theta_1\circ\xi(\overline{d})$, hence $\sigma_1 = mgu(\phi(a'),\overline{d})$ exists and for some $\psi\in SUBST$: $\psi\circ\sigma_1 = \theta_2\circ\theta_1\circ\xi$. We also have $\psi\circ\sigma_1\circ\phi(a) = \theta_2\circ\theta_1\circ\xi\circ\phi(a) = \theta_2\circ\theta_1\circ\phi(a) = \theta_2\circ\theta_1(d) = \theta_2\circ\theta_1\circ\xi(d') = \theta_2\circ\theta_1\circ\xi\circ\phi(\overline{d}') = \psi\circ\sigma_1\circ\phi(\overline{d}')$, hence $\sigma_2 = mgu(\sigma_1\circ\phi(a),\overline{d}')$ exists and for some $\psi'\in SUBST$: $\psi = \psi'\circ\sigma_2$. So the existence of g_1' and g_2' is proven.

$\sigma_2\circ\sigma_1\circ\xi\circ\phi(a) = \sigma_2\circ\sigma_1\circ\phi(a) = \sigma_2(\overline{d}') = \sigma_2\circ\sigma_1(\overline{d}') = \sigma_2\circ\sigma_1\circ\xi(d)$. Hence for some $\chi\in SUBST$: $\chi\circ\theta_1 = \sigma_2\circ\sigma_1\circ\xi$. Also $\chi\circ\theta_1\circ\phi(a') = \sigma_2\circ\sigma_1\circ\xi\circ\phi(a') = \sigma_2\circ\sigma_1\circ\phi(a') = \sigma_2\circ\sigma_1(\overline{d}) = \sigma_2\circ\sigma_1\circ\xi(\overline{d}) = \chi\circ\theta_1(d')$. So for some $\chi'\in SUBST$: $\chi'\circ\theta_2 = \chi$.

Now we have $\psi'\circ\sigma_2\circ\sigma_1 = \theta_2\circ\theta_1\circ\xi$ and $\chi'\circ\theta_2\circ\theta_1 = \sigma_2\circ\sigma_1\circ\xi$. Let $\eta=\psi'|_{rge(\sigma_2\circ\sigma_1)}$ and $\zeta=\chi'|_{rge(\theta_2\circ\theta_1)}$. Then $\eta\circ\sigma_2\circ\sigma_1 = \theta_2\circ\theta_1\circ\xi$ and $\zeta\circ\theta_2\circ\theta_1 = \sigma_2\circ\sigma_1\circ\xi$. From $\sigma_2\circ\sigma_1 = \sigma_2\circ\sigma_1\circ\xi\circ\xi = \zeta\circ\theta_2\circ\theta_1\circ\xi = \zeta\circ\eta\circ\sigma_2\circ\sigma_1$ we derive $\zeta(\eta(x))=x$ for each $x\in rge(\sigma_2\circ\sigma_1)$ hence $\zeta\circ\eta=id$. So η is a renaming and $\eta\circ\sigma_2\circ\sigma_1\circ\xi = \theta_2\circ\theta_1$.

We have $\theta_2\circ\theta_1\circ\phi(c) = \eta\circ\sigma_2\circ\sigma_1\circ\xi\circ\phi(c) = \eta\circ\sigma_2\circ\sigma_1\circ\phi(c)$, $\theta_2\circ\theta_1\circ\phi(\alpha) = \eta\circ\sigma_2\circ\sigma_1\circ\xi\circ\phi(\alpha) = \eta\circ\sigma\circ\sigma\circ\xi(a) = \eta\circ\sigma_2\circ\sigma_1(\overline{\alpha}')$ and $\theta_2\circ\theta_1\circ\phi(\alpha') = \eta\circ\sigma_2\circ\sigma_1\circ\xi\circ\phi(\alpha') = \eta\circ\sigma_2\circ\sigma_1\circ\xi(\alpha') = \eta\circ\sigma_2\circ\sigma_1(\overline{\alpha})$. So $\theta_2\circ\theta_1\circ\phi(c_2) = \eta\circ\sigma_2\circ\sigma_1\circ\phi(c_2')$. □

(2.14) PROPOSITION Let $\Delta=(g_i)_{i=0}^{n}$ be a derivation from $<c,\phi,m>$. Let $g_l=<c_l,\phi_l,m+l>$ and a,a' be conjuncts of c_l. Suppose $g_l\rightarrow g_{l+1}$ by resolution of $\phi_l(a)$ against $\alpha\in P_{m+l+1}$ via mgu θ_{l+1} and $g_{l+1}\rightarrow g_{l+2}$ by resolution of $\theta_{l+1}\circ\phi_l(a')$ against $\alpha'\in P_{m+l+2}$ via mgu θ_{l+2}. Let $\xi: VAR_{m+l+1}\cup VAR_{m+l+2} \rightarrow VAR_{m+l+1}\cup VAR_{m+l+2}$ be canonical. Let $\overline{\alpha}=\xi(\alpha')$, $\overline{\alpha}'=\xi(\alpha)$. Then

there exists a derivation $\Delta'=(g_i')_{i=0}^n$ from $<c,\phi,m>$ such that
1. $g_{i-1}'\rightarrow g_i'$ via mgu σ_i, $(1\leqslant i\leqslant n)$;
2. $g_l'\rightarrow g_{l+1}'$ by resolution of $\phi_l(a')$ against $\bar{\alpha}$ via mgu σ_{l+1};
3. $g_{l+1}'\rightarrow g_{l+2}'$ by resolution of $\sigma_{l+1}\circ\phi_l(a)$ against $\bar{\alpha}'$ via mgu σ_{l+2};
4. $[\Delta(c)]=[\Delta'(c)]$.

PROOF Take $g_i'=g_i$, $(1\leqslant i\leqslant l)$. Take g_{l+1}',g_{l+2}' as in proposition (2.13). Let η be a renaming such that $index(\eta)\leqslant m+l+2$, $\eta\circ\sigma_{l+2}\circ\ ..\ \circ\sigma_1\circ\phi(c_{l+2}')=\theta_{l+2}\circ..\circ\theta_1\circ\phi(c_{l+2})$ and $\theta_{l+2}\circ\theta_{l+1}\circ\xi=\eta\circ\sigma_{l+2}\circ\sigma_{l+1}$. Proposition (2.12)(3) guarantees the existence of g_i', $(l+3\leqslant i\leqslant n)$ and of a renaming ζ such that $\theta_n\circ..\circ\theta_{l+3}\circ\eta=\zeta\circ\sigma_n\circ..\circ\sigma_{l+3}$. Finally, note that $\theta_n\circ..\circ\theta_1\circ\phi(c)=\zeta\circ\sigma_n\circ\ ..\ \sigma_{l+3}\circ\eta\circ\theta_{l+2}\circ\ ..\ \circ\theta_1\circ\phi(c)=\zeta\circ\sigma_n\circ\ ..\ \circ\sigma_{l+1}\circ\xi\circ\theta_l\circ\ ..\ \circ\theta_1\circ\phi(c)=\zeta\circ\sigma_n\circ\ ..\ \circ\sigma_{l+1}\circ\xi\circ\sigma_l\circ\ ..\ \sigma_1\circ\phi(c)=\zeta\circ\sigma_n\circ\ ..\ \circ\sigma_1\circ\phi(c)$. so for $\Delta=(g_i')_{i=0}^n$ we have $[\Delta(c)]=[\Delta'(c)]$. □

(2.15) THEOREM (Independence of the computation-rule): Let $g=<c,\psi,m>\in GOAL$. Let ρ be a computation-rule such that there exists a successful derivation from g according ρ. Then there exists for every computation-rule ρ' a successful derivation from g according ρ'. Moreover if ϕ and ϕ' are the results of the resp. computations then $\phi(c)$ is a variant of $\phi'(c)$.

PROOF By the above proposition. □

EQUIVALENCE OF FIXPOINT SEMANTICS AND OPERATIONAL SEMANTICS.

In this section we establish agreement of the fixpoint semantics and the operational one. At forehand there is a problem, because the two semantics relate different kinds of objects with a logic program, viz. a set of ground atoms versus a mapping from formulae to sets of substitution and bottom. Our way out is the notion of success set.

(2.16) DEFINITION The success set $Suc_\emptyset(P)$ of a logic program P is defined by $Suc_\emptyset(P)=\{\ a\in HB\mid$ there exists a successful derivation from $a\ \}$.

In order to settle the equivalence of $\mathcal{F}(P)$ and $Suc_\emptyset(P)$ we first prove

(2.17) LEMMA Let P be a logic program. Let a be a ground atom in $\mathcal{T}^n(\emptyset)$. Then $a\in Suc_\emptyset(P)$.

PROOF Induction on n. Case n=1: Let $a\in\mathcal{T}(\emptyset)$. For some axiom $a'\leftarrow$**true** in P and substitution $\phi,\phi(a')=a$. Let $\bar{a}\leftarrow$**true** be the variant of $a'\leftarrow$**true** in P_1. For a suitable renaming γ we have $\phi(\gamma(a))=a=\phi(a')=\phi(\gamma(\bar{a}))$. So $\theta=mgu(a,\bar{a})$ exists and $a\rightarrow<$**true**$,\theta,1>$. Hence $a\in Suc_\emptyset(P)$.
Case n+1←n: Let $a\in\mathcal{T}^{n+1}(\emptyset)$. Without loss of generality, for some rule $a_0'\leftarrow a_1'\wedge..\wedge a_r'\in$P, $(r\geqslant 1)$ and substitution ϕ, $\phi(a_0')=a$ and $\phi(a_1'),..,\phi(a_r')\in\mathcal{T}^n(\emptyset)$. Let $\bar{a}_0\leftarrow\bar{a}_1\wedge..\wedge\bar{a}_r$ be the variant of $a_0'\leftarrow a_1'\wedge..\wedge a_r'$ in P_1. Then $\theta=mgu(a,a_0')$ exists and for a suitable renaming γ and substitution ψ we have $\psi\circ\theta=\phi\circ\gamma$. Since $\phi(\gamma(\bar{a}_1))$,.., $\phi(\gamma(\bar{a}_r))\in\mathcal{T}^n(\emptyset)$ we know from the induction hypothesis that $\wedge_{i=1}^r\phi(\gamma(\bar{a}_i))$ produces **true**. By the rectangle lemma it follows that $<\wedge_{i=1}^r a_i,\theta,1>$ produces **true**. We conclude $a\in Suc_\emptyset(P)$. □

(2.18) COROLLARY Let P be a logic program and a a ground atom. If $a\in\mathcal{F}(P)$, then $a\in Suc_\emptyset(P)$. Hence $\mathcal{F}(P)\subseteq Suc_\emptyset(P)$.

PROOF Immediate by the lemma, because $lfp(\mathcal{T})=\cup_{n=o}^\infty\mathcal{T}^n(\emptyset)$. □

The above corollary settles in some sense the completeness of the operational framework. The next lemma is preparatory to the soundness of the operational semantics.

(2.19) LEMMA Let P be a logic program, and $c=\wedge_{i=1}^s a_i$, $(s\geqslant 1)$ a conjunction of index m. Suppose $c\rightarrow^*<$**true**$,\phi,m+n>$. Then $[\phi(a_i)]\subseteq\mathcal{T}^n(\emptyset)$, $1\leqslant i\leqslant s$.

PROOF Induction on n. Case n=1: trivial. Case n+1←n: say $<\wedge_{i=1}^s a_i,id,m>\rightarrow<\wedge_{i=1}^j a_i\ \wedge\ \wedge_{i=1}^r a_i'\ \wedge\ \wedge_{i=j+1}^s a_i,\ \theta,m+1>\rightarrow^*<$**true**$,\psi\circ\theta,m+n+1>$ with $\phi=\psi\circ\theta$. We distinguish three cases.

(i). $k \neq j$: So a_k is not replaced in the first step. By the induction hypothesis for $\theta(a_k)$ we have $[\phi(a_k)] \subseteq [\psi(\theta(a_k))] \subseteq \mathcal{T}^n(\varnothing) \subseteq \mathcal{T}^{n+1}(\varnothing)$.
(ii). a_j is replaced by resolution against an axiom: $[\phi(a_j)] \subseteq [\theta(a_j)] \subseteq \mathcal{T}(\varnothing)$.
(iii). a_j is replaced by resolution against a rule: $[\phi(a_i')] = [\psi(\theta(a_i'))] \subseteq \mathcal{T}^n(\varnothing)$, $1 \leqslant i \leqslant r$ by the induction hypothesis. Hence $[\phi(a_j)] \subseteq \mathcal{T}^{n+1}(\varnothing)$ by the definition of $\mathcal{T}$. □

(2.20) THEOREM (1). Let P be a logic program and a a ground atom. If $a \in Suc_\theta(P)$, then $a \in F(P)$.
(2). For every logic program P: $Suc_\theta(P) = \mathcal{F}(P)$.

PROOF Immediate by the lemmas. □

LEFTMOST-SEMANTICS.

(2.21) DEFINITION Let P be a logic program. We define the binary relation
$_P\rightarrow_l$ on $GOAL$ as the least one such that $<\wedge_{i=1}^r a_i, \phi, m>$
$_P\rightarrow_l$ $<\wedge_{i=1}^s a_i' \wedge \wedge_{i=2}^r a_i, \theta \circ \phi, m+1>$ if some clause $a'_0 \leftarrow a'_1 \wedge .. \wedge a'_r \in P_{m+1}$, $\theta = mgu(\phi(a_1), a'_0)$ exists.
$_P\rightarrow_l^*$ is the transitive closure of $_P\rightarrow_l$.

(2.22) DEFINITION The leftmost-semantics $\mathcal{L}(P)$ of a logic program P is defined by $\mathcal{L}(P): CONJ \rightarrow \mathcal{P}(SUBST \cup \{\perp\})$ such that for each $c \in CONJ$: $\phi \in \mathcal{L}(P)(c)$, $\phi \neq \perp$ iff for some $n \geqslant 1$ $c\ _P\rightarrow_l^* <\mathbf{true}, \phi, n>$, and $\perp \in \mathcal{L}(P)(c)$ iff $c\ _P\rightarrow_l^* \Omega$.

(2.23) DEFINITION Define the success set $Suc_{\mathcal{L}}(P)$ with respect to leftmost derivations of a logic program P as $Suc_{\mathcal{L}}(P) = \{ a \in HB \mid$ there exists a successful leftmost derivation from $a \}$.

(2.24) THEOREM Let P be a logic program. Then $Suc_\theta(P) = Suc_{\mathcal{L}}(P)$.

PROOF Immediate by theorem (2.15). □

DENOTATIONAL SEMANTICS.

We have already defined the model-theoretic, fixpoint-, operational and leftmost-semantics of logic programs. In this section we will define a denotational one. From denotational point of view a logic program is just syntactical object, built up from its components. Moreover, a logic program without input is regarded as meaningless, similar to an imperative program with procedure declarations only. We will treat logic programming as execution of a body, (the input) in presence of a set of procedures, (the logic program). Furthermore, we want the definition of a denotational semantics to be compositional, i.e. the meaning of a logic program with input should be given by the meaning of its parts.

We consider the set $BIN = \{ (\phi, n) \in SUBST \times NUM \mid index(\phi) \leqslant n \} \cup \{\perp\}$ as a discrete c.p.o. and the set $\mathcal{P}_{EM}(BIN) = \{ \sigma \in \mathcal{P}(ENV) \mid \sigma$ finite, or infinite and containing $\perp \}$ as a c.p.o. with the Egli-Milner ordering. We introduce the set $ENV = ATOM \rightarrow (BIN \rightarrow_s \mathcal{P}_{EM}(BIN))$. ENV is a c.p.o. with the ordering induced by $BIN \rightarrow_s \mathcal{P}_{EM}(BIN))$.

We will use β, σ, γ as typical elements of BIN, $\mathcal{P}_{EM}(BIN)$ and ENV resp. So $\beta_1 \leqslant \beta_2$ iff either $\beta_1 = \perp$, or $\beta_1 = \beta_2$; $\sigma_1 \leqslant \sigma_2$ iff either $\perp \in \sigma_1$ and $\sigma_1 - \{\perp\} \subseteq \sigma_2$, or $\sigma_1 = \sigma_2$; $\gamma_1 \leqslant \gamma_2$ iff for all a and β: $\gamma_1(a)(\beta) \leqslant \gamma_2(a)(\beta)$.

An environment is supposed to give the possible result of a leftmost computation starting with an atom a under some assignment ϕ. With this in mind we know how to deal with conjunctions. If we start up the program with **true** as input, then there is nothing to do. If an atom a is input of the program, then we just inspect the environment. If we have a conjunction $c_1 \wedge c_2$ as input, then we first process c_1 and process c_2 afterwards, taking account of the side-effects caused by c_1.

(2.25) DEFINITION The mapping $\mathcal{B}: CONJ \rightarrow ENV \rightarrow (BIN \rightarrow \mathcal{P}_{EM}(BIN))$ is defined by
1. $\mathcal{B}(\mathbf{true})(\gamma) = \lambda\beta.\{\beta\}$;

2. $\mathfrak{B}(a)(\gamma)=\gamma(a)$;
3. $\mathfrak{B}(c_1 \wedge c_2)(\gamma) = \mathfrak{B}(c_2)(\gamma) \circ \mathfrak{B}(c_1)(\gamma)$.

To give meaning to a logic program with certain input we just have to know which environment we should create. So we define a transformation of environments, invoke our order-theoretic machinery to find the least fixpoint of this transformation, and claim that this environment is the one we search for.

(2.26) DEFINITION The transformation $\Phi: PROG \to (ENV \to ENV)$ is defined by $\Phi(\alpha)(\gamma)(a)(\phi,n) = \cup \{ \mathfrak{B}(c')(\gamma)(\theta \circ \phi, n+1) \mid index(a) \leqslant n,\ a' \leftarrow c' \in \alpha_{n+1},\ \theta = mgu(\phi(a),a') \}$,
$\Phi(P' \cup P'')(\gamma)(a) = \Phi(P')(\gamma)(a) \cup \Phi(P'')(\gamma)(a)$.

(2.27) LEMMA For all $c \in CONJ$: $\mathfrak{B}(c) \in ENV \to (BIN \to_s \mathcal{P}_{EM}(BIN))$ is continuous. For all $P \in PROG$: $\Phi(P) \in ENV \to ENV$ is continuous.

PROOF Standard. □

From the lemma we deduce that for each program P the transformation $\Phi(P)$ has a least fixpoint. So the next definition is justified.

(2.28) DEFINITION The denotational semantics $\mathfrak{D}(P \leftarrow c)$ of a sentence P←c, i.e. a logic program with input is defined as $\mathfrak{D}: SENT \to \mathcal{P}_{EM}(BIN)$, $\mathfrak{D}(P \leftarrow c) = \mathfrak{B}(c)(\gamma)(id,0)$ where $\gamma = lfp(\Phi(P \leftarrow c))$.

EQUIVALENCE OF LEFTMOST- AND DENOTATIONAL SEMANTICS.

Let P be a logic program and $\gamma = lfp(\Phi(P))$. Put $\gamma_0 = \lambda a.\lambda\beta.\{\ \bot\ \}$ and $\gamma_{k+1} = \Phi(P)(\gamma_k)$. We write Φ instead of $\Phi(P)$.

(2.29) LEMMA Let $\beta=(\phi,n)$, $\beta'=(\phi',n') \in BIN$. Suppose $\beta' \in \mathfrak{B}(c)(\gamma_k)(\beta)$. Then $<c,\phi,n> \to_l^* <\mathbf{true},\phi',n'>$.

PROOF Induction on k and the structure of c. We only treat two cases.
Case $k+1 \leftarrow k, c=a$: $\beta' \in \mathfrak{B}(a)(\gamma_{k+1})(\beta)$ iff $\beta' \in \Phi(\gamma_k)(a)(\beta)$ iff there exists $a' \leftarrow c' \in P_{n+1}$ such that $\theta = mgu(\phi(a),a')$ exists, $index(a) \leqslant n$, $\beta' \in \mathfrak{B}(c')(\gamma_k)(\theta \circ \phi, n+1)$. By induction $<c',\theta \circ \phi, n+1> \to_l^* <\mathbf{true},\phi',n'>$. So $<a,\phi,n> \to_l^* <\mathbf{true},\phi',n'>$.
Case $k+1 \leftarrow k, c=c_1 \wedge c_2$: $\beta' \in \mathfrak{B}(c_1 \wedge c_2)(\gamma_{k+1})(\beta)$ iff $\beta' \in \mathfrak{B}(c_2)(\gamma_{k+1})(\bar{\beta})$ where $\bar{\beta} \in \mathfrak{B}(c_1)(\gamma_{k+1})(\beta)$. By induction: $<c_2,\bar{\phi},\bar{n}> \to_l^* <\mathbf{true},\phi',n'>$, and $<c_1,\phi,n> \to_l^* <true,\bar{\phi},n>$. Hence $<c_1 \wedge c_2,\phi,n> \to_l^* <\mathbf{true},\phi',n'>$. □

(2.30) LEMMA Let $<c,\phi,n> \in GOAL$. Suppose $<c,\phi,n> \to_l^* <\mathbf{true},\phi',n'>$. Let $\beta=(\phi,n)$ and $\beta'=(\phi',n')$. Then $\beta' \in \mathfrak{B}(c)(\gamma)(\beta)$.

PROOF Induction on $k=n'-n$. Case $k+1 \leftarrow k, c=a$: $<a,\phi,n> \to_l <c',\theta \circ \phi, n+1> \to_l^* <\mathbf{true},\phi',n'>$. By induction $\beta' \in \mathfrak{B}(c')(\gamma)(\theta \circ \phi, n+1)$ and $\mathfrak{B}(c')(\gamma)(\theta \circ \phi) \subseteq \mathfrak{B}(a)(\gamma)(\phi)$. Hence $\beta' \in \mathfrak{B}(a)(\gamma)(\beta)$.
Case $k+1 \leftarrow k, c=c_1 \wedge c_2$: $<c_1 \wedge c_2,\phi,n> \to_l^* <c_2,\bar{\phi},\bar{n}> \to_l^* <\mathbf{true},\underline{\phi}',n'>$. By induction: $\bar{\beta} \in \mathfrak{B}(c_1)(\gamma)(\beta)$ and $\beta' \in \mathfrak{B}(c_2)(\gamma)(\bar{\beta})$. Hence $\beta' \in \mathfrak{B}(c_1 \wedge c_2)(\gamma)(\beta)$ where $\bar{\beta}=(\phi,\bar{n})$. □

(2.31) LEMMA Let $<c,\phi,n> \in GOAL$. If $\bot \notin \mathfrak{B}(c)(\gamma_k)(\phi)$, then there is no infinite chain out of $<c,\phi,n>$, i.e. $<c,\phi,n> \to_l^* \Omega$ is not the case.

PROOF Induction on k and c. Case $k+1 \leftarrow k, c=c_1 \wedge c_2$: Suppose $\bot \notin \gamma_{k+1}(c_1 \wedge c_2)(\beta)$. Hence $\bot \notin \gamma_{k+1}(c_1)(\beta)$. and $\bot \notin \gamma_{k+1}(c_2)(\beta')$ for every $\beta' \in \mathfrak{B}(c_1)(\gamma_{k+1})(\beta)$. Thus by induction: every chain out of $<c_1,\phi,n>$ has an end and for every $\beta' \in \mathfrak{B}(c_1)(\gamma_{k+1})(\beta)$ every chain out of $<c_2,\phi',n'>$ has an end. Thus $<c_1 \wedge c_2,\phi,n>$ begins no infinite chain. □

(2.32) LEMMA If $\bot \in \mathfrak{B}(c)(\gamma_k)(\beta)$ then $<c,\phi,n> \to_l^* <c',\phi',n'>$ with $n' \geqslant n+k$.

PROOF Induction on k and c. Notice $<c,\phi,n> \in GOAL$. Case $k+1 \leftarrow k, c=a$: Suppose $\bot \in \mathfrak{B}(a)(\gamma_{k+1})(\beta)$. Then $\bot \in \Phi(P)(\gamma_k)(a)(\beta)$, hence $\bot \in \mathfrak{B}(c')(\gamma_k)(\theta \circ \phi, n+1)$ where $a' \leftarrow c' \in P_{n+1}$

and $\theta = mgu(\phi(a), a')$. By induction hypothesis for $<c', \theta \circ \phi, n+1>$ we are ready.
Case $k+1 \leftarrow k, c = c_1 \wedge c_2$: Suppose $\perp \in \mathfrak{B}(c_1 \wedge c_2)(\gamma_{k+1})(\beta)$. So $\perp \in \mathfrak{B}(c_1)(\gamma_{k+1})(\beta)$ or $\perp \in \mathfrak{B}(c_2)(\gamma_{k+1})(\beta')$ with $\beta' \in \mathfrak{B}(c_1)(\gamma)(\beta)$, hence $<c_1, \phi, n> \rightarrow_l^* <\bar{c}, \phi, \bar{n}>$ with $\bar{n} \geqslant n+k+1$ or $<c_1, \phi, n> \rightarrow_l^* <\mathbf{true}, \phi', n'>$ and $<c_2, \phi', n'> \rightarrow_l^* <\bar{c}, \phi, \bar{n}>$ with $\bar{n} \geqslant n+k+1$. Hence $<c_1, \phi, n> \rightarrow_l^* <\bar{c}, \phi, \bar{n}>$ with $\bar{n} \geqslant n+k+1$. □

(2.33) COROLLARY If $\perp \in \mathfrak{B}(c)(\gamma)(\beta)$, then $<c, \phi, n> \rightarrow_l^* \Omega$.

PROOF From the above lemma and the one due to König. □

(2.34) DEFINITION Define the success set $Suc_{\mathfrak{D}}(P)$ by $Suc_{\mathfrak{D}}(P) = \{ a \in HB \mid \mathfrak{D}(P \leftarrow a) - \{\perp\} \neq \varnothing \}$.

(2.35) THEOREM For each sentence $P \leftarrow a$: $\mathfrak{D}(P \leftarrow a) = \mathfrak{L}(P)(a)$ and $Suc_{\mathfrak{L}}(P) = Suc_{\mathfrak{D}}(P)$. □

2.2. Another approach

In case we want to know if different derivations in the operational model deliver the same substitutions, we can do this by counting them.
In this section we associate with a sentence a multiset of answer substitutions. This approach gives us some more information about the program than the previous one. For example one can determine when in the program a rule is defined twice or that a set of rules gives the same answers as an other set of rules does. We restrict us to the leftmost computation rule.

SOME MATHEMATICAL DEFINITIONS AND PRELIMINARIES

Let again $BIN = \{ (\phi, n) \in (SUBST \times NUM) \mid index(\phi) \leqslant n \} \cup \{ \bot \}$. Let NUM^∞ stands for the set of natural numbers with infinity (∞).
We define: $\mathcal{P}_{mul}(BIN) = \{M : BIN \to NUM^\infty \mid M(\bot) = 0 \text{ or } M(\bot) = \infty \}$ and
$MUL = \{M \in \mathcal{P}_{mul}(BIN) \mid M \text{ is finite} \} \cup \{ M \in \mathcal{P}_{mul}(BIN) \mid M(\bot) = \infty \}$.
It follows that $\bot$ is in M or it is not in M; the multiplicity is of no interest.

The following notations will be used:
- $M \in \mathcal{P}_{mul}(BIN)$ *is finite* iff $\Sigma_{\beta \in BIN} M(\beta) < \infty$
- $M = \{\beta_i / n_i \mid i \in I \}$ means $\forall i \in I : M(\beta_i) = n_i$ and $\forall \beta \notin \{ \beta_i \mid i \in I \} : M(\beta) = 0$.

We will use MUL as our domain for the semantics of the operational and denotational semantics. Because we use multisets we have to adapt some operators on sets and we introduce a new operator $\oplus$ (called sum).

(2.36) DEFINITION Let $M_1, M_2 \in \mathcal{P}_{mul}(BIN)$ and $(M_i)_{i \in NUM}$ be a collection of sets from $\mathcal{P}_{mul}(BIN)$.
1. $M_1 \cup M_2 := \lambda\beta.max\{ M_1(\beta), M_2(\beta) \}$
2. $\cup_{i \in I} M_i := \lambda\beta.sup \{ M_i(\beta) \mid i \in I \}$
3. $M_1 \oplus M_2 := \lambda\beta.M_1(\beta) + M_2(\beta)$
4. $\oplus_{i \in I} M_i := \lambda\beta.\Sigma_{i \in I} M_i(\beta)$

We define the inclusion relation and an Egli-Milner like ordering on MUL. It will be shown that with this ordering MUL is a c.p.o..

(2.37) DEFINITION 1. Let $M_1, M_2 \in \mathcal{P}_{mul}(BIN)$. $M_1 \subseteq_{mul} M_2$ iff $\forall a \in domain(M_1)$: $M_1(a) \leqslant M_2(a)$
2. Let $M_1, M_2 \in MUL$. $M_1 \leqslant M_2$ iff either $M_1(\bot) = 0$ and $M_1 = M_2$ or $M_1(\bot) = \infty$ and $\forall \beta \in BIN - \{ \bot \} : M_1(\beta) \leqslant M_2(\beta)$.

Remarks that it is the case that $M_1 \cup M_2$ and $M_1 \oplus M_2$ are elements of $\mathcal{P}_{mul}(BIN)$ and so are $\cup_{i \in I} M_i$ and $\oplus_{i \in I} M_i$. If M_1 and M_2 are elements of MUL then $M_1 \oplus M_2$ and $M_1 \cup M_2$ are elements of MUL. It is generally not the case that $<M_i>_i$ is a collection of sets in MUL implies $\cup_{i=0}^{\infty} M_i \in MUL$. If however $<M_i>_i$ is a chain in MUL then $\cup_{i=0}^{\infty} M_i \in MUL$.

(2.38) THEOREM $(MUL, \leqslant, \{ \bot / \infty \})$ is a cpo.

PROOF We only define and check the least upper bound ($\sqcup$). Let $<M_i>_i$ be a chain in MUL. If there exists i_0 such that $\forall i \geqslant i_0 : M_i = M_{i_0}$ then take $\sqcup_{i=0}^{\infty} M_i = M_{i_0}$. If no such i_0 exists, take $\sqcup_{i=0}^{\infty} M_i = \cup_{i=0}^{\infty} M_i$. We check that $\cup_{i=0}^{\infty} M_i$ in the second case is indeed the least upperbound of $<M_i>_i$.
Clearly for all i: $M_i \leqslant \cup_{i=0}^{\infty} M_i$, because for all i: $(\bot, \infty) \in M_i$. If $M' \in MUL$ is an upperbound of $<M_i>_i$. Let $\beta \in BIN$, then $\forall i \geqslant 0 : M_i(\beta) \leqslant M'(\beta)$, thus $sup \{M_i(\beta) \mid i \geqslant 0 \} \leqslant M'(\beta)$, thus $\cup_{i=0}^{\infty} M_i(\beta) \leqslant M'(\beta)$. Hence $\cup_{i=0}^{\infty} M_i \leqslant M'$ since $(\bot, \infty) \in \cup_{i=0}^{\infty} M_i$. $\square$

For technical reasons we change the definition of $CONJ$, which will still be a sequence of atoms but now parenthesized in a specific way. We introduce $BODY$ which is a conjunction of at least one atom or an empty conjunction, denoted by **true**.

The class *CONJ*, with typical elements *c*,..., is defined by: $c ::= a \mid a \wedge c$.
The class *BODY*, with typical elements *b*,..., is defined by: $b ::=$ **true** $\mid c$.
We are now ready to define the operational semantics with multisets as our domain.

(2.39) DEFINITION The *operational semantics* $\mathcal{O}: SENT \rightarrow MUL$ is defined by:
$\mathcal{O}(P \leftarrow b) = \oplus \{ \{(\phi, n)\} \mid b \ {}_P\!\rightarrow^* (\textbf{true}, \phi, n) \} \cup$ **if** $b \ {}_P\!\rightarrow^* \Omega$ **then** $\{(\perp, \infty)\}$ **else** $\varnothing$ **fi**.

We define some more operators and prove their continuity.

(2.40) DEFINITION We define $\otimes : NUM^\infty \times \mathcal{P}_{mul}(BIN) \rightarrow \mathcal{P}_{mul}(BIN)$ by
$n \otimes M = \{ \beta \ / \ (n \times M(\beta)) \mid \beta \in BIN \} \cup$ **if** $n = \infty$ **then** $\{ \perp \ / \ \infty \}$ **else** $\varnothing$ **fi**
From the definition we have: if $M \in MUL$ then $n \otimes M \in MUL$. This is due to the fact that in case $n = \infty$ we include bottom.

(2.41) LEMMA The operators $\oplus$ and $\cup$ are monotonic in both arguments and $\otimes$ is monotonic in its second argument with respect to the $\leqslant$ ordering. $\otimes$ is monotonic in its first argument with respect to the adapted inclusion relation.

PROOF Omitted. □

(2.42) PROPOSITION If $<M_i>_i$ is a chain in *MUL* then
1. $\square_{i=0}^{\infty} M_i = \{ \beta \ / \ \lim_{i\rightarrow\infty} M_i(\beta)) \mid \beta \neq \perp \} \cup \{ \perp \ / \ \infty \mid$ if $\forall i: M_i(\perp) = \infty \}$
2. If $\forall i: M_i \subseteq_{mul} M_{i+1}$ then $\cup_{i=0}^{\infty} M_i = \lambda\beta. \lim_{i\rightarrow\infty} M_i(\beta)$
Let $(M_i)_i$ and $(N_i)_i$ be collections of sets of *MUL*. Then:
3. $\forall i: M_i \subseteq_{mul} N_i \rightarrow \oplus_{i=0}^{\infty} M_i \subseteq_{mul} \oplus_{i=0}^{\infty} N_i$
4. $\forall i: M_i \leqslant N_i$ and $\exists i: N_i(\perp) = \infty \rightarrow \oplus_{i=0}^{\infty} M_i \leqslant \oplus_{i=0}^{\infty} N_i$

PROOF the proof is straightforward. □
Let $<n_i>_i$ be a chain in NUM^∞.

(2.43) LEMMA The operators $\oplus$ and $\cup$ are continuous in both arguments and $\otimes$ is continuous in its second argument with respect to the $\leqslant$ ordering. For $\otimes$ holds: $((lim_{i\rightarrow\infty} n_i) \otimes M) \mid_{BIN-\{\perp\}} = \cup_{i=0}^{\infty} (n_i \otimes M) \mid_{BIN-\{\perp\}}$

PROOF Depends on the fact that $\lim_{i\rightarrow\infty} (m_i + n) = (\lim_{i\rightarrow\infty} m_i) + n$ and also for $\times$ and *max*. □

(2.44) DEFINITION Let $t_1, t_2: BIN \ {}_s\!\rightarrow MUL$ then the composition $t_2 \circ t_1$ of t_1 and t_2 is defined by: $t_2 \circ t_1 = \lambda\beta. \oplus \{ n' \otimes t_2(\beta') \mid (\beta', n') \in t_1(\beta) \}$

Note that $\circ$ will always deliver an element of *MUL* because of strictness of the functions t_i.
If t_1 is finite and $\forall\beta$: ($t_1(\beta) \neq 0 \rightarrow t_2(\beta)$ is finite) then $t_2 \circ t_1$ is finite. In all the other cases bottom is an element of $t_2 \circ t_1$.
With this definition and the previous lemmas we can prove the following.

(2.45) LEMMA $\circ$ is monotonic and continuous in both arguments.

PROOF The proof depends on the previous proposition and the fact that $\Sigma_{j\in J} \lim_{i\rightarrow\infty} m_{j,i} = \lim_{i\rightarrow\infty} \Sigma_{j\in J} m_{j,i}$, if $<m_{j,i}>_i$, is an increasing chain in NUM^∞. □

THE DENOTATIONAL SEMANTICS WITH MULTISETS

The class *ENV*, with typical elements γ,..., is defined by:
$ENV = ATOM \rightarrow (BIN \ {}_s\!\rightarrow MUL)$. *ENV* is a cpo because *MUL* is, and it has as its bottom element $\lambda\, a\, (\phi, n).\{\perp\}$.
Now we are ready to define the denotational semantics.

(2.46) DEFINITION The mapping $\mathcal{B}: CONJ \rightarrow ENV \rightarrow (BIN \rightarrow_s MUL)$ is defined by
1. $\mathcal{B}(\textbf{true})(\gamma) = \lambda\beta.$ **if** $\beta \neq \perp$ **then** $\{\beta / 1\}$ **else** $\{ \perp \ / \ \infty \}$ **fi** ;
2. $\mathcal{B}(a)(\gamma) = \gamma(a)$;

3. $\mathfrak{B}(a \wedge c)(\gamma) = \mathfrak{B}(c)(\gamma) \circ \mathfrak{B}(a)(\gamma)$.

(2.47) DEFINITION The operator $\Phi: PROG \rightarrow (ENV \rightarrow ENV)$ is defined by
1. $\Phi(a' \leftarrow b')(\gamma)(\phi,n) = \oplus \{ B(b')(\gamma)(\theta \circ \phi, n+1) \mid \theta = mgu(\phi(a), \psi_{n+1}(a')) \text{ exists} \}$;
2. $\Phi(P_1 \cup P_2) = \lambda\gamma a \lambda\beta . \Phi(P_1)(\gamma)(a)(\beta) \oplus \Phi(P_2)(\gamma)(a)(\beta)$.

(2.48) DEFINITION The mapping $\mathfrak{D}: SENT \rightarrow MUL$ is defined by $\mathfrak{D}(P \leftarrow b) = \mathfrak{B}(b)(\gamma)(id,0)$ with $\gamma = lfp(\Phi(P))$.

(2.49) THEOREM 1. Let $b \in BODY$ and $\beta \in BIN$. Then $\lambda\gamma . B(b)(\gamma)(\beta)$: $ENV \rightarrow MUL$ is continuous.
2. Let $P \in PROG$. Then $\Phi(P)$: $ENV \rightarrow ENV$ is continuous. □

We will not prove the equivalence of the last defined operational and denotational semantics. It is obvious that the operational semantics with multisets differs only from the one in section (2.1) with respect to multiplicity. It also is not difficult to prove that the two denotational semantics are, modulo multiplicity, equivalent.
In the next chapter we give another operational model with multisets as semantic domain.

Chapter 3

Logic programming, Prolog and Concurrent Prolog

INTRODUCTION

In this chapter we give an operational model of logic programming with multisets. This model is not only a preparatory study of Concurrent Prolog, (Shapiro, [45]), but also an introduction to the techniques used in the models for Prolog and Concurrent Prolog. In the second part of this chapter we give an operational and denotational semantics of a subset of Prolog, as described in an article of Jones and Mycroft [26]. We prove the equivalence of them. The last part of this chapter is based on the article of Shapiro [45], which describes Concurrent Prolog. We add an operational model of CP.

3.1. The operational semantics of logic programming

In the previous chapter we looked at derivations. A derivation corresponds to one path in the search tree. Now we take a more global point of view by looking at the whole search tree.

The classes *ATOM*, *CONJ*, *BODY*, *CLAUSE*, *PROG* and *SENT* are defined as before.

Let X be a set, then X^* denotes the set of finite sequences of elements of X. ϵ denotes the empty sequence.

The class *STACK*, with typical elements s,..., is defined by: $STACK = BODY^*$

The class *STATE*, with typical elements σ,..., is defined by:

$\sigma \in STATE$ iff $\sigma \in (STACK \times SUBST \times SENT \times NUM)$ or $\sigma ::= \sigma_1 \square \sigma_2$ and $\sigma_1, \sigma_2 \in STATE$.

We define some transition rules (our axioms) with disjoint domains. Further we need some inference rules, which give us the possibility to generate a derivation.

The transition relation "$\rightarrow$" is defined from $STATE \times \mathcal{P}_{mul}(BIN)$ to $STATE \times \mathcal{P}_{mul}(BIN) \cup \mathcal{P}_{mul}(BIN)$.

We define the start goal by $<$(b, Id, P, 0), $\varnothing>$.

P is a set of definite clauses, which defines the whole program. P', P_1 and P_2 are subsets of P.

THE DEFINITION OF THE TRANSITION RULES:

$$<(a \bullet s, \phi, a' \leftarrow b', n), W> \rightarrow <(\psi_{n+1}(b') \bullet s, (\theta \circ \phi), P, n+1), W> \qquad (1)$$

$$\textit{if } \theta = mgu(\phi(a), \psi_{n+1}(a')) \textit{ exists.}$$

$$\rightarrow W \textit{ otherwise.}$$

$$<((a \wedge c) \bullet s,\ \phi,\ a' \leftarrow b',\ n),\ W> \rightarrow <(\psi_{n+1}(b') \bullet c \bullet s,\ (\theta \circ \phi),\ P,\ n+1),\ W> \quad (2)$$

if $\theta = mgu(\phi(\ a\ ),\ \psi_{n+1}(\ a'\ ))$ *exists.*

$\rightarrow$ W *otherwise.*

$$<(\mathbf{true} \bullet s,\ \phi,\ P',\ n),\ W> \rightarrow <(s,\ \phi,\ P',\ n),\ W> \quad (3)$$

$$<(\epsilon,\ \phi,\ P',\ n),\ W> \rightarrow W \oplus \{\ (\phi, n)\ /\ 1\ \} \quad (4)$$

$$(c \bullet s,\ \phi,\ P_1 \,\|\, P_2,\ n),\ W> \rightarrow <(c \bullet s,\ \phi,\ P_1,\ n) \,\square\, (c \bullet s,\ \phi,\ P_2,\ n),\ W> \quad (5)$$

THE DEFINITION OF THE INFERENCE RULES:

$$\frac{<\sigma,\ W> \rightarrow <\sigma',\ W'>}{<\sigma \,\square\, \sigma_1,\ W> \rightarrow <\sigma' \,\square\, \sigma_1,\ W'>} \quad (6)$$

$$\frac{<\sigma,\ W> \rightarrow\ W'}{<\sigma \,\square\, \sigma_1,\ W> \rightarrow <\sigma_1,\ W'>} \quad (7)$$

$$\frac{<\sigma,\ W> \rightarrow <\sigma',\ W'>}{<\sigma_1 \,\square\, \sigma,\ W> \rightarrow <\sigma_1 \,\square\, \sigma',\ W'>} \quad (8)$$

$$\frac{<\sigma,\ W> \rightarrow\ W'}{<\sigma_1 \,\square\, \sigma,\ W> \rightarrow <\sigma_1,\ W'>} \quad (9)$$

A production sequence is a finite or infinite sequence of the form:
(*) $<\sigma_0,\ W_0>, <\sigma_1,\ W_1>, \ldots,\ W_n$, or
(**) $<\sigma_0,\ W_0>, <\sigma_1,\ W_1>, \ldots, <\sigma_n,\ W_n>, <\sigma_{n+1},\ W_{n+1}>, \cdots$
where $<\sigma_n,\ W_n> \rightarrow <\sigma_{n+1},\ W_{n+1}>$ holds for $n = 0, 1, 2, \cdots$.
Let $\perp$ be defined as before. We say that $<\sigma,\ W> \rightarrow^{\infty} W'$, where $W' \in MUL$. $\rightarrow^{\infty}$, may be pronounced as "produces finitely or infinitely many steps", whenever one of the following conditions is satisfied.

1. There is a finite sequence (*) with $<\sigma_0, W_0> = <\sigma,\ W>$ and $<\sigma_n,\ W_n> \rightarrow\ W'$.
2. There is an infinite sequence (**) with $<\sigma_0, W_0> = <\sigma,\ W>$ and the sequence $<W_n>_n$ is not constant and $W' = (\ \cup_{i=0}^{\infty} W_i)\ \cup\ \{\ \perp\ /\ \infty\ \}$
3. There is an infinite sequence (**) with $<\sigma_0, W_0> = <\sigma,\ W>$ and the sequence $<W_n>_n$ is constant for $n \geqslant n_0$ for some n_0 and $W' = W_{n_0}\ \cup\ \{\ \perp\ /\ \infty\ \}$.

We define $\vartheta : SENT \rightarrow MUL$ by
$\vartheta(P \leftarrow b) = \cup\ \{\ W' \in MUL \mid <(b,\ Id,\ P,\ 0),\ \varnothing> \rightarrow^{\infty} W'\ \}$

SOME COMMENTS ON THE DEFINITIONS ABOVE

In rule (1) and (2) we see that the leftmost atom is replaced by the body of the clause $a' \leftarrow b'$, if the most general unifier exists. Rule (3) is a way of skipping the empty clause. Rule (4) collects the answer substitutions. Rule (5) takes care of the unification of the leftmost atom in the goal with *all* the clauses of the program. This rule together with the inference rules has as consequence, that all possible derivations are computed simultaneously (as discussed in the introduction).
In condition 2. we include $\perp$, because $\perp$ denotes the presence of an unfinished computation and an infinite production sequence trivially denotes an unfinished computation.

In the next sections, whenever a b with a ϕ and an index n and some more information occurs together, the following constraints hold: $index(b) \leqslant n$, $index(\phi) \leqslant n$.

SOME LEMMAS

In order to prove that $\Theta(P \leftarrow b)$ is an element of *MUL*, we define $\Theta(b,m,\phi,P,n) = \cup \{ W \mid <(b,\phi,P,n), \varnothing> \rightarrow^{\infty} W \}$ with $index(b) = m$, and prove that $\Theta(b,m,\phi,P,n)$ is an element of *MUL*. Since $\Theta(b, 0, Id, P, 0) = \Theta(P \leftarrow b)$ it follows that $\Theta(P \leftarrow b) \in MUL$.

(3.1) LEMMA All production sequences of $<(q,\phi,P,n), \varnothing>$ are finite iff the leftmost search tree of (q,ϕ,P,n) is finite.

PROOF Omitted. □

(3.2) LEMMA If one production sequence that starts with $<(b,\phi,P,n), \varnothing>$ is finite then all the production sequences are finite.

PROOF The difference between two production sequences is the choice between which σ_i you choose to develop, if the state is $\sigma_1 \square \cdots \square \sigma_n$ and σ_i's are of the form (s_i,ϕ_i,P_i,n_i). Because we have a finite sequence, this means in the end we have no state left, every σ_i will be developed eventually, so there is no state which will lead to an infinite production sequence. □

(3.3) THEOREM $\Theta(b,m,\phi,P,n) \in MUL$.

PROOF $<(b,\phi,P,n), \varnothing> \rightarrow^{\infty} W$.

i) $(\perp, \infty) \notin W$ (i.e. W is finite). then $W = \Theta(b,m,\phi,P,n)$, because we know that all production sequences are finite, $(\perp, \infty) \notin \Theta(b,m,\phi,P,n)$.

Let $((\theta,p),k) \in \Theta(b,m,\phi,P,n)$ When we have in our production sequence state $\sigma_1 \square \cdots \square \sigma_n$, defined as above. Then the number of (θ, p) is in every W the same, because the only difference between two production sequences is the choice of σ_i and in the last step of the production sequence state disappears (because all the production sequences are finite). So every σ_i has been developed. Thus $((\theta,p),k) \in W$ and $W = \Theta(b,m,\phi,P,n)$

ii) $(\perp, \infty) \in W$ then $\forall W'$: $<(b,\phi,P,n), \varnothing> \rightarrow^{\infty} W'$: $(\perp, \infty) \in W'$ and thus $(\perp, \infty) \in \Theta(b,m,\phi,P,n)$. □

THE EQUIVALENCE OF THE OPERATIONAL AND DENOTATIONAL SEMANTICS

In this section we will prove that for each program P and each goal b the operational and denotational semantics coincide.

We will prove that $\Theta(b, m, \phi, P, n) = B(b)(\gamma)(\phi, n)$ with $\gamma = lfp\Phi(P)$.

First we prove some useful lemmas.

(3.4) LEMMA If $((\theta, p), s) \in \Theta(b, m, \phi, P, n)$ then $s < \infty$

PROOF Every state is of the form: $<\sigma_1 \square \cdots \square \sigma_k, W>$ with $\sigma_i = (s_i, \phi_i, P_i, n_i)$. As soon as $n_i > p$ the σ_i is of no interest to us (we refer to transition rule (4)).

Let us assume that all $n_i \leqslant p$.

The only way to increase the n_i's is by applying transition rules (1) and (2).

The only way to increase the number of σ_i's is to use rule (5), but the amount of new elements of the form of the σ_i's is equal to number of clauses in the program which is finite, so for every state $k < \infty$.

In a finite number of steps we can create a state in which all the n_i's are equal to p. So the total number of (θ, p)'s is also finite, which means that $s < \infty$. □

(3.5) LEMMA Let $\beta \neq \perp$. If $(\beta, s) \in \Theta(b, m, \phi, P, n)$, then there is a "minimal" production sequence (finite or infinite) $<(b,\phi,P,n), \varnothing>, <\sigma_1, W_1>, \cdots$ such that $(\beta, s) \in W_l$ and $(\beta, s) \notin W_{l-1}$ and for all other production sequences, whenever $(\beta, s) \in W_k$ then $k \geqslant l$.

PROOF Omitted. □

In the above situation l is called *the minimal production index* of $(\beta\ s)$.

(3.6) DEFINITION $MPI_l(\ b,\ m,\ \phi,\ P,\ n) = \{\ (\beta,\ s) \in \Theta(\ b,\ m,\ \phi,\ P,\ n)\ \mid$ *the minimal production index of* $(\beta,\ s)$ *is* $l\ \}$

We write MPI_l for short , if no confusion is possible.
It is obvious that $MPI_0 = \varnothing$ and $\Theta(\ b,\ m,\ \phi,\ P,\ n)\ |_{BIN-\{\perp\}} = \cup_{i=0}^{\infty} MPI_i$
If we write $(\ \theta,\ p,\ s)$ we mean $((\ \theta,\ p),\ s)$

(3.7) LEMMA $(\ \theta,\ p,\ s) \in \Theta(\ a \wedge c,\ m,\ \phi,\ P,\ n)$ if and only if $\forall i = 1, \ldots, l\ (\ \theta_i,\ p_i,\ s_i) \in \Theta(\ a,\ m,\ \phi,\ P,\ n)$ such that $(\ \theta,\ p,\ t_i) \in \Theta(\ c,\ m,\ \theta_i \circ \phi,\ P,\ p_i)$ and $\Sigma_{i=1}^{l}\ s_i \times t_i = s$

PROOF Omitted. □

(3.8) DEFINITION $\Psi_p(\ b,\ m,\ \phi,\ P,\ n) = B(b)\ (\Phi(P)^p(\lambda\ a\ \phi\ n.\{\perp\}))\ (\phi,\ n)$

(3.9) LEMMA $\Theta(\ b,\ m,\ \phi,\ P,\ n)\ |_{BIN-\{\perp\}} = \Box_{p=0}^{\infty}\ \Psi_p(\ b,\ m,\ \phi,\ P,\ n)\ |_{BIN-\{\perp\}}$

PROOF $\subseteq_{mul}$) With induction on the minimal production index and the complexity of b. We prove:
$\forall l\ \forall b\ :(\theta,\ k,\ s) \in MPI_l\ \ \exists p_b\colon\ \ \forall p \geqslant p_b\colon (\theta,\ k,\ s) \in \Psi_p(\ b,\ m,\ \phi,\ P,\ n)$, as follows: If $l = 0$ then nothing has to be proved and we are ready.
So let it be proven for minimal production sequences less than l.
- $b \equiv$ **true**; $p_b = 0$ and we are ready.
- $b \equiv a$; $\Theta(\ a,\ m,\ \phi,\ P,\ n) = \Theta(\ b_1,\ n+1,\ \theta_1 \circ \phi,\ n+1)\ \oplus \cdots \oplus\ \Theta(\ b_t,\ n+1,\ \theta_t \circ \phi,\ n+1)$, with $a_i \leftarrow b_i$ in P and $\theta_i = mgu(\ \phi(a),\ \psi_{n+1}(a_i))$ exists.
The minimal production index of $(\ \theta,\ k,\ s_i) \in\ \Theta(\ b_i,\ n+1,\ \theta_i \circ \phi,\ n+1)$ is less than the one of $\Theta(\ a,\ m,\ \phi,\ P,\ n)$, with $\Sigma_{i=1}^{t}\ s_i = s$.
So $\forall p \geqslant p_{b_i}\colon (\ \theta,\ k,\ s_i) \in\ \Psi_p(b_i,\ n+1,\ \theta_i \circ \phi,\ P,\ n+1)$.
Let $p_{\max} = max\{\ p_{b_i}\ \mid\ i = 1, \ldots, t\ \}$
then $\forall p \geqslant p_{\max}\colon\ (\ \theta,\ k,\ s) \in\ \Psi_p(\ b_1,\ n+1,\ \theta_1 \circ \phi,\ n+1)\ \ \oplus .. \oplus\ \ \Psi_p(\ b_t,\ n+1,\ \theta_t \circ \phi,\ n+1) = \Psi_{p+1}(\ a,\ m,\ \phi,\ P,\ n)$
- $b \equiv a \wedge c$; $(\theta,\ k,\ s) \in \Theta(a \wedge c,\ m,\ \phi,\ P,\ n)$ then with induction and the previous lemma we have: $\forall i = 1, \ldots, l\ \ \forall p \geqslant p_{a,i}\colon\ (\ \theta_i,\ p_i,\ s_i) \in\ \ \Psi_p(\ a,\ m,\ \phi,\ P,\ n)$ such that $\forall p \geqslant p_{c,i}\colon\ (\ \theta,\ p,\ t_i) \in \Psi_p(\ c,\ m,\ \theta_i \circ \phi,\ P,\ p_i)$
Let $p_{\max} = max\{\ p_{a,i}\ ,\ p_{c,j}\ \mid\ i = 1, \ldots, l$ and $j = 1, \ldots, l\ \}$
then $\forall p \geqslant p_{\max}\colon\ (\ \theta,\ k,\ \Sigma_{i=1}^{l}\ s_i \times t_i)\ \ \in \oplus_{i=1}^{l}\ (s_i \otimes\ \ \Psi_p(\ c,\ m,\ \theta_i \circ \phi,\ p_i))$ thus $\forall p \geqslant p_{\max}\colon (\ \theta,\ k,\ s) \in \Psi_p(\ a \wedge c,\ m,\ \phi,\ n)$
$\supseteq_{mul}$) to prove: $\forall p\colon\ \forall b\colon \Psi_p(\ b,\ m,\ \phi,\ P,\ n)\ \leqslant \Theta(\ b,\ m,\ \phi,\ P,\ n)$. Induction on the complexity of b and p and previous lemma. Further proof omitted. □

(3.10) LEMMA $(\perp,\ \infty) \notin \Theta(\ b,\ m,\ \phi,\ P,\ n)$ iff $(\perp,\ \infty) \notin \Box_{p=0}^{\infty}\ \Psi_p(\ b,\ m,\ \phi,\ P,\ n)$

PROOF *only if)* With induction on b and the length of the production sequence. Since for all W such that $<(b,m,\phi,P,n),\ \varnothing> \rightarrow_{\infty} W$ we have $W = \Theta(\ b,\ m,\ \phi,\ P,\ n))$.
Assume that the production sequence has non-zero length.
- $b \equiv$ **true** ; then we are ready.
- $b \equiv a$; We take the following production sequence:
$<(a,\phi,P,n),\ \varnothing>, \ldots, <\sigma_1\ \Box \cdots\ \Box\ \sigma_t,\ \varnothing>$, where $\sigma_i = (b_i,\ \theta_i \circ \phi,\ P,\ n+1)$, and $a_i \leftarrow b_i$ in P and $\theta_i = mgu(\ \phi(a),\ \psi_{n+1}(a_i))$ exist, $1 \leqslant i \leqslant t$.
It is true that $\Theta(\ a,\ m,\ \phi,\ P,\ n) = \Theta(b_1,\ n+1,\ \theta_1 \circ \phi,\ P,\ n+1) \oplus \cdots \oplus \Theta(b_t,\ n+1,\ \theta_t \circ \phi,\ P,\ n+1)$ and the length of the production steps of the b_i's are shorter than a, so $(\perp,\ \infty) \notin \Box_{p=0}^{\infty}\ \Psi_p(b_i,\ n+1,\ \theta_i \circ \phi,\ P,\ n+1)$ and $\Box_{p=0}^{\infty}\ \Psi_p(\ a,\ m,\ \phi,\ P,\ n) =$
$\Box_{p=0}^{\infty}\ \oplus_{i=1}^{t}\ \Psi_p(b_i,\ n+1,\ \theta_i \circ \phi,\ P,\ n+1) = \oplus_{i=1}^{t} \Box_{p=0}^{\infty}\ \Psi_p(b_i,\ n+1,\ \theta_i \circ \phi,\ P,\ n+1)$,
so $(\perp,\ \infty) \notin \Box_{p=0}^{\infty}\ \Psi_p(\ a,\ m,\ \phi,\ P,\ n)$.
- $b \equiv a \wedge c$; if $(\perp,\ \infty) \notin \Theta(\ a \wedge c,\ m,\ \phi,\ P,\ n)$, then $(\perp,\ \infty) \notin \Theta(\ a,\ m,\ \phi,\ P,\ n)$ and $(\perp,\ \infty) \notin \Theta(\ c,\ m,\ \theta \circ \phi,\ P,\ k)$: $\forall(\theta,\ k,\ s) \in \Theta(\ a,\ m,\ \phi,\ P,\ n)$, so $(\perp,\ \infty) \notin \Box_{p=0}^{\infty}\ \Psi_p(\ a,\ m,\ \phi,\ P,\ n)$ and $\forall(\theta,\ k,\ s) \in \Box_{p=0}^{\infty}\ \Psi_p(\ a,\ m,\ \phi,\ P,\ n)$:

$(\perp, \infty) \notin \square_{p=0}^{\infty} \Psi_p(c, m, \theta \circ \phi, P, k)$ and thus $(\perp, \infty) \notin \square_{p=0}^{\infty} \Psi_p(a \wedge c, m, \phi, P, n)$

if) $(\perp, \infty) \notin \square_{p=0}^{\infty} \Psi_p(b, m, \phi, P, n)$ this means there is a p such that $(\perp, \infty) \notin \Psi^p(b, m, \phi, P, n)$. Proof with induction on b and p, it is an analogous proof as the *"only if" case* part as described above. Further proof is omitted. □

Now we have that $\Theta(b, m, \phi, P, n) = \square_{p=0}^{\infty} \Psi_p(b, m, \phi, P, n)$ and this implies $\mathcal{O}(P \leftarrow b) = \mathcal{D}(P \leftarrow b)$. With these last two lemmas we have proved the equivalence of the operational and denotational semantics.

3.2. The operational and denotational semantics of a subset of Prolog

Prolog is a logic programming language first developed by Alain Colmerauer of the University of Marseille in 1970, [44]. The Prolog interpreter uses as computation rule leftmost and search rule depth first (with an ordering on the clauses).

INTRODUCTION

We know that different computation rules lead to the same results. However different search rules can lead to different results. For example: Consider the program $P := \{\ n(0) \leftarrow \textbf{true},\ n(s(x_1)) \leftarrow n(x_1)\ \}$ and the goal $\leftarrow n(x_1)$.
Take as search rule:

in each node take the left most son not yet visited and mark this node as visited, if none is left, go up one node and repeat this procedure.

With this search rule we find all the natural numbers. But if we have as search rule:

in each node take the right most son not yet visited and mark this node as visited, if none is left, go up one node and repeat this procedure.

then we will find no answer substitutions.

In this section a operational and a denotational semantics of Prolog as proposed by Jones and Mycroft, [26] is given. We add an equivalence proof of these two semantics.

SOME MATHEMATICAL DEFINITIONS AND PRELIMINARIES

Before we can give the operational- and denotational semantics we have to develop a domain and some operators on it.
Let BIN now stand for $SUBST \times NUM$, $\perp$ is a new element not in BIN. We define BIN^{tr} as $BIN^{tr} := BIN^* \cup BIN^* \perp \cup BIN^\infty$.
The following ordering can be defined on BIN^{tr}. If x and y are elements of BIN^{tr}, then $x \leqslant y$ if either $x \in BIN^* \cup BIN^\infty$ and $x = y$, or $x = x' \perp$ and x' is a prefix of y. In [3] and [37] it is proven that $(BIN^{tr}, \leqslant, \perp)$ is a cpo.

We have a concatenation operator $'\bullet'$ defined as follows: $x \bullet y = xy$ if $x \in BIN^*$, and $x \bullet y = x$ if $x \in BIN^* \perp \cup BIN^\infty$. $'\bullet'$ is continuous in both arguments. See [3].

THE OPERATIONAL SEMANTICS OF PROLOG

The definition of the classes $BODY$, $CONJ$, $CLAUSE$ and $SENT$ are not changed. The class $PROG$ is slightly modified. It is defined by: $P ::= \alpha \mid \alpha \bullet P$
Because the Prolog interpreter uses the so-called backtracking mechanism it has to update control information at each visit of a node in the search tree. This is done by stack management. The whole stack forms the state of the derivation. Each element of the stack contains information about the current subgoal, the clauses which we have not yet applied to the subgoal, the substitution so far generated and a renaming index for the clauses.

We define the class $STACK$, with typical elements s,.. by $STACK = BODY^*$. We define the class $STATE$, with typical elements σ,.. by $STATE = (\ STACK \times SUBST \times PROG \times NUM\ )^*$.
We define a transition relation $" \rightarrow "$ on $STATE \times BIN^{tr}$ to $(\ STATE \times BIN^{tr}\ ) \cup BIN^{tr}$. For a sentence $P \leftarrow b$, we define the start goal by $<(b, Id, P, 0), \epsilon>$.

THE DEFINITION OF THE TRANSITION RULES:

$$<(a \bullet s,\ \phi,\ (a' \leftarrow b') \bullet P',\ n) \bullet \sigma,\ w> \tag{1}$$

$$\rightarrow <(\psi_{n+1}(b') \bullet s,\ (\theta \circ \phi),\ P,\ n+1) \bullet \sigma',\ w>$$

if $\theta = mgu(\phi(a), \psi_{n+1}(a'))$ *exists.*

$$\rightarrow <\sigma',\ w> \quad \textit{otherwise.}$$

with $\sigma' = (a \bullet s,\ \phi,\ P',\ n) \bullet \sigma$

$$<((a \wedge c) \bullet s,\ \phi,\ (a' \leftarrow b') \bullet P',\ n) \bullet \sigma,\ w> \tag{2}$$

$$\rightarrow <(\psi_{n+1}(b') \bullet c \bullet s,\ (\theta \circ \phi),\ P,\ n+1) \bullet \sigma',\ w>$$

if $\theta = mgu(\phi(a), \psi_{n+1}(a'))$ *exists.*

$$\rightarrow <\sigma',\ w> \quad \textit{otherwise.}$$

with $\sigma' = ((a \wedge c) \bullet s,\ \phi,\ P',\ n) \bullet \sigma$

$$<(\textbf{true} \bullet s,\ \phi,\ P',\ n) \bullet \sigma,\ w> \rightarrow <(s,\ \phi,\ P,\ n) \bullet \sigma,\ w> \tag{3}$$

$$<(c \bullet s,\ \phi,\ \epsilon,\ n) \bullet \sigma,\ w> \rightarrow <\sigma,\ w> \tag{4}$$

$$<(\epsilon,\ \phi,\ P',\ n) \bullet \sigma,\ w> \rightarrow <\sigma,\ w \bullet (\phi, n)> \tag{5}$$

$$<\epsilon,\ w> \rightarrow w \tag{6}$$

Again a production sequence is a finite or infinite sequence of the form:
(*) $<\sigma_0, w_0>, <\sigma_1, w_1>, \ldots, w_n$, or
(**) $<\sigma_0, w_0>, <\sigma_1, w_1>, \ldots, <\sigma_n, w_n>, <\sigma_{n+1}, w_{n+1}>, \ldots$
where $<\sigma_n, w_n> \rightarrow <\sigma_{n+1}, w_{n+1}>$ holds for $n = 0, 1, 2, \ldots$ We say that $<\sigma, w> \rightarrow^{\infty} w'$, where $w' \in BIN^{tr}$, whenever one of the following conditions is satisfied:

1. There is a finite sequence (*) with $<\sigma_0, w_0> = <\sigma, w>$ and $<\sigma_n, w_n> \rightarrow w'$.
2. There is an infinite sequence (**) with $<\sigma_0, w_0> = <\sigma, w>$ and the sequence $<w_n>_n$ is constant for $n \geq n_0$ for some n_0 and $w' = w_{n_0} \bullet \perp$.
3. There is an infinite sequence (**) with $<\sigma_0, w_0> = <\sigma, w>$ and the sequence $<w_n>_n$ is not constant and $w' = sup_n\ w_n$.

We define: $\mathcal{O}(P \leftarrow b) = w$, if $<(b, Id, P, 0), \epsilon> \rightarrow^{\infty} w$.

Some remarks: It must be noticed that in every production step exactly one of the transition rules can be chosen. In transition rules (1) and (2) we resolve a against $a' \leftarrow b'$. Because we want to apply the rules in P' on a too, we save that information on the stack (see σ'). Rule (4) simulates the so called backtracking mechanism of Prolog. If no rules are left to unify with, the top element of the stack is popped. In case we have reduced our goal to ϵ, (rule (5)), the answer substitution is delivered and the top element is popped of the stack. It is obvious that the w_n's in the production sequence are prefixes of the w_{n+1}'s, so $sup_n\ w_n$ in condition 3. is well defined.

SOME MATHEMATICAL DEFINITIONS AND PRELIMINARIES

Before we give a description of the denotational semantics, we must define a new function:

(3.11) DEFINITION *append* : $((BIN \rightarrow BIN^{tr}) \times BIN^{tr}) \rightarrow BIN^{tr}$ is defined by

$$append(t, x) = \begin{cases} t(a) \bullet append(t, x') & \text{if } x = ax' \text{ with } x' \in BIN^{*}\perp \cup BIN^{*} \\ x & \text{if } x = \epsilon \text{ or } x = \perp \\ \Box_{i=0}^{\infty} append(t, x_i) & \text{if } x \in BIN^{\infty} \text{ with } <x_i>_i \text{ a chain} \\ & \text{and } \forall i: x_i \in BIN^{*}\perp \text{ and } x = \Box_{i=0}^{\infty} x_i \end{cases}$$

(3.12) LEMMA *append* is monotonic and continuous in both arguments.

PROOF With induction on the length of x. □

THE DENOTATIONAL SEMANTICS

Again we have the class *ENV* of environments with typical elements γ,.. defined by $ENV = ATOM \times BIN \rightarrow BIN^{tr}$. *ENV* is a cpo because BIN^{tr} is, with bottom element: $\lambda\, a\, (\phi, n)\, .\, \perp$.

(3.13) DEFINITION The mapping $\mathfrak{B} : BODY \times ENV \times BIN \rightarrow BIN^{tr}$ is defined by:

1. $\mathfrak{B}(\mathbf{true})(\gamma)(\phi,n) = (\phi,n)$;
2. $\mathfrak{B}(a)(\gamma)(\phi,n) = \gamma(a)(\phi,n)$;
3. $\mathfrak{B}(a \wedge c)(\gamma)(\phi, n) = append(\lambda\, (\theta,n').\ \mathfrak{B}(c)(\gamma)((\theta \circ \phi),n'),\ \gamma(a)(\phi,n))$.

(3.14) DEFINITION The function $\Phi : PROG \rightarrow (ENV \rightarrow ENV)$ is defined by:

1. $\Phi(\epsilon)(a)(\phi,n) = \epsilon$;
2. $\Phi((a' \leftarrow b') \bullet P')(\gamma)(a)\ \ (\phi,n) = \sigma \bullet \sigma'$, with $\sigma = \mathfrak{B}(\psi_{n+1}(b'))\ \ (\gamma)((\theta \circ \phi),(n+1))$ if $\theta = mgu(\phi(a),\psi_{n+1}(a'))$ exists, else $\sigma = \epsilon$ and $\sigma' = \Phi(P')(\gamma)(a)(\phi,n)$

(3.15) DEFINITION The function $\mathfrak{D} : SENT \rightarrow (BIN)^{tr}$ is defined by: $\mathfrak{D}(P \leftarrow b) = \mathfrak{B}(b)(\gamma)(Id, 0)$ with: $\gamma = lfp\ \Phi(P) = \Box_{p=0}^{\infty} (\Phi(P))^{p}(\lambda a\, \beta.\, \perp)$.

With this definition and the previous lemmas we can prove the following theorem:

(3.16) THEOREM 1. $\lambda\, \gamma.\mathfrak{B}(b)\ (\gamma)\ (\phi,\ n) : ENV \rightarrow BIN^{tr}$ is a continuous function

2. $\Phi(P) : ENV \rightarrow ENV$ is a continuous function

PROOF 1. With induction on the complexity of b and the continuity of *append*.

2. With induction on the complexity of P and the continuity of $\mathfrak{B}$ and $\bullet$. □

THE EQUIVALENCE OF THE OPERATIONAL AND DENOTATIONAL SEMANTICS

In this section we prove that for each program P and for each goal b, the operational and denotational semantics coincide. First some definitions.

(3.17) DEFINITION Let $index(b) = m$. We define:

1. $\Theta(b, m, \phi, P, n) = w$ iff $<(b, \phi, P, n), \epsilon> \rightarrow^{\infty} w$.
2. $\Psi_p(b, m, \phi, P, n) = B(b)\ (\Phi(P)^{p}\ (\lambda\, a\ (\phi, n)\, .\, \perp))\ (\phi,\ n)$.
3. $\Psi(b, m, \phi, P, n) = \Box_{p=0}^{\infty} \Psi_p(b, m, \phi, P, n)$.

Notice that $\Box_{p=0}^{\infty} \Psi_p(b, m, \phi, P, n) = B(b)fix\ (\Phi(P))\ (\phi, n)$. We will prove $\forall b, m, \phi, P, n$: $\Theta(b, m, \phi, P, n) = \Box_{p=0}^{\infty} \Psi_p(b, m, \phi, P, n)$ Before we can prove this there are several propositions to be provided. Most of their proves are straight forward with induction on the complexity of b or the length of the production sequence or on the index p.

(3.18) PROPOSITION If $\square_{p=0}^{\infty} \Psi_p(b, m, \phi, P, n) = (\theta_1, n_1)\bullet(\theta_2, n_2)\bullet (\theta_3, n_3)\bullet..$ then $\forall i: n_i \geqslant n, \theta_i \in SUBST$ and $\theta_i = \theta_i' \circ \phi$.

PROOF Omitted. □

A similar proposition holds for $\Theta(b, m, \phi, P, n)$.

(3.19) PROPOSITION Let $<\sigma_1, \epsilon>, <\sigma_2, \epsilon>,..$ be a production sequence. with $\sigma_1 = (b, \phi, P, n)$, then $\forall i: \sigma_i = (s_1^i, \phi_1^i, P_1^i, n_1^i)\bullet (s_2^i, \phi_2^i, P_2^i, n_2^i)\bullet.. (s_{l_i}^i, \phi_{l_i}^i, P_{l_i}^i, n_{l_i}^i)$
$\forall i, j: \phi_j^i \in SUBST$ and $\phi_j^i = \theta_j^i \circ \phi$.

PROOF With induction on the length of the production sequence. □

(3.20) PROPOSITION The following two assertions are equivalent:

(1) $<(a, \phi, P, n), \epsilon>, <\sigma_2, \epsilon> ,.., <\sigma_p, \epsilon>, <\sigma_{p+1}, (\phi_1^p, n_1^p)>, \sigma_i = (s_1^i, \phi_1^i, P_1^i, n_1^i)\bullet (s_2^i, \phi_2^i, P_2^i, n_2^i) \bullet..\bullet (a, \phi, P', n)$, with $s_1^p = \epsilon$ and $s_j^i \neq \epsilon$,

(2) $<(a \wedge c, \phi, P, n), \epsilon>, <\sigma_2', \epsilon> ,.., <\sigma_p', \epsilon>, \sigma_i' = (s_1^i \bullet c, \phi_1^i, P_1^i, n_1^i)\bullet (s_2^i \bullet c, \phi_2^i, P_2^i, n_2^i) \bullet..\bullet (a \bullet c, \phi, P', n)$, with $s_1^p = \epsilon$ and $s_j^i \neq \epsilon$.

PROOF with induction on the length of the production sequence. □

(3.21) PROPOSITION The following two assertions are equivalent:

(1) $< \sigma_1 \bullet \sigma_2 \bullet.. \bullet \sigma_k, w_n> ,.., < \sigma_2 \bullet.. \bullet \sigma_k, w_{n+p}>$,
(2) $< \sigma_1, w_n> ,.., < \epsilon, w_{n+p}>$,

PROOF Omitted. □

(3.22) PROPOSITION Let $<(a, \phi, P, n), \epsilon> \rightarrow^{\infty} (\theta_1, n_1)\bullet(\theta_2, n_2) \bullet(\theta_3, n_3)\bullet..$. Then $\forall i: <(c, \theta_i \circ \phi, P, n_i), \epsilon> \rightarrow^{\infty} (\phi_{i,1}, n_{i,1})\bullet(\phi_{i,2}, n_{i,2}) \bullet(\phi_{i,3}, n_{i,3})\bullet..$, if and only if $<(a \wedge c, \phi, P, n), \epsilon> \rightarrow^{\infty} (\phi_{1,1}, n_{1,1})\bullet(\phi_{1,2}, n_{1,2}) \bullet..\bullet (\phi_{2,1}, n_{2,1})\bullet(\phi_{2,2}, n_{2,2}) \bullet..\bullet (\phi_{3,1}, n_{3,1})\bullet(\phi_{3,2}, n_{3,2}) \bullet..$.

PROOF with all the previous propositions. If a production sequence is infinite for certain i, then substitutions with index higher than i disappear. □

(3.23) LEMMA $\forall p \ \forall b: \Psi_p(b, m, \phi, P, n) \leqslant \Theta(b, m, \phi, P, n)$

PROOF With induction on p and the complexity of b.

$b \equiv a$: $\Psi_p(a, m, \phi, P, n) = \Psi_{p-1}(b_1, n+1, \theta_1 \circ \phi, P, n+1) \bullet..\bullet \Psi_{p-1}(b_t, n+1, \theta_t \circ \phi, P, n+1)$ with $a_i \leftarrow b_i$ occuring in P in that order and $\theta_i = mgu(\phi(a), \psi_{n+1}(a_i))$ exists. Our induction hypothesis says: $\Psi_{p-1}(b_i, n+1, \theta_i \circ \phi, P, n+1) \leqslant \Theta(b_i, n+1, \theta_i \circ \phi, P, n+1)$. Since $\Theta(a, m, \phi, P, n) = \Theta(b_1, n+1, \theta_1 \circ \phi, P, n+1) \bullet..\bullet \Theta(b_t, n+1, \theta_t \circ \phi, P, n+1)$, it follows that $\Psi_p(a, m, \phi, P, n) \leqslant \Theta(a, m, \phi, P, n)$.
$b \equiv a \wedge c$: $\Psi_p(a \wedge c, m, \phi, n) = append(\lambda\, \theta\, n'.\ \Psi_p(c, m, (\theta \circ \phi), n'), \Psi_p(m, a, \phi, n))$. The induction hypothesis gives us: $\forall(\theta, n')$: $\Psi_p(c, m, (\theta \circ \phi), n') \leqslant \Theta(c, m, (\theta \circ \phi), n')$, and $\Psi_p(a, m, \phi, n) \leqslant \Theta(a, m, \phi, n)$. Then with the previous proposition, it follows:
$append(\lambda\, \theta\, n'.\ \Theta(c, m, (\theta \circ \phi), n'), \Theta(a, m, \phi, n)) = \Theta(a \wedge c, m, \phi, n)$, so $\Psi_p(a \wedge c, m, \phi, n) \leqslant \Theta(a \wedge c, m, \phi, n)$. □

(3.24) DEFINITION We define

$$\Theta_k(b, m, \phi, P, n) = \begin{cases} w & \begin{cases} \text{if } <(b, m, \phi, P, n)\ \epsilon>, <\sigma_1, w_1>,..,<\sigma_{k-1}, w_{k-1}>, w \\ \text{is its production sequence.} \end{cases} \\ w_k \perp & \begin{cases} \text{if } <(b, m, \phi, P, n)\ \epsilon>, <\sigma_1, w_1>,..,<\sigma_k, w_k>,.. \\ \text{is its production sequence.} \end{cases} \end{cases}$$

(3.25) LEMMA $\square_{k=0}^{\infty} \Theta_k(b, m, \phi, P, n) = \Theta(b, m, \phi, P, n)$

PROOF Omitted. □

(3.26) LEMMA $\forall k \ \forall b: \ \Theta_k(b, m, \phi, P, n) \leqslant \Psi(b, m, \phi, P, n)$

PROOF With induction on k and b. An analogous proof as the previous lemma. □
Finally we have arrived at:

(3.27) THEOREM $\mathfrak{O}(P \leftarrow b) = \mathfrak{P}(P \leftarrow b)$. □

3.3. · The operational semantics of Concurrent Prolog

In this part of our paper we will describe an operational model of Concurrent Prolog. CP is a variant of the programming language Prolog. Prolog is a sequential simulation of a parallel computation model. Shapiro gives in [45] two reasons for exploiting Prolog's underlying parallelism: One is to improve the performance of Prolog in some of its current applications, perhaps using novel computer architectures. The other is to incorporate in the range of Prolog applications that require concurrency. CP is concerned with both.

In the same article Shapiro gives a sketch of a CP interpreter. We will closely follow this sketch to define our operational model. But before we come to the definition of the transition- and inference rules, we describe the syntax and the informal semantics of CP.

CP adds two syntactic constructs to logic programs: *read-only annotation* of variables, e.g. $x?$, and the *commit operator* ":", (Shapiro uses "|".) Both are used to control the computation, i.e. the construction of the derivation, by restricting the order in which goals can be resolved and restricting the choice of clauses that can be used to resolve them. A CP-program is a set of guarded clauses. A guarded clause is a universally quantified axiom of the form:

$$A \leftarrow G_1 \wedge G_2 \wedge .. \wedge G_m : B_1 \wedge .. \wedge B_n, \quad m, n \geqslant 0$$

where the G's and the B's are atoms. The G's form the guard of the clause. When the guard is empty the commit operator is omitted. The clause may contain variables marked "read-only".

The commit operator generalizes and cleans up the cut operator of sequential Prolog. Declaratively it reads like a conjunction. A is implied by G's $\wedge$ B's. Operationally a guarded clause and an atom can be resolved against each other if the head of the clause and the atom are unifiable and the guard of the clause can be resolved successfully. The bindings made during the evaluating of the guard have to be composed with the associated substitution.

The unification of terms containing read-only variables is an extension of the normal unification. The unification of a read-only variable $x?$ with a term t succeeds only if t is a variable, (hence not a read-only variable). If in an atom a a read-only variable $x?$ occurs and in the associated substitution ϕ, x receives a value not equal to a variable, then the annotation of the read-only variable $x?$ disappears in $\phi(a)$.

We represent the read-only annotations as a unary functor written in postfix notation. The unification algorithm will handle this functor in a special manner, as described above. We notice that the read-only annotation is an annotation strictly bounded to the variable it has as its argument. This means that if $x?$ is a read-only variable and x is assigned a non-variable term which contains at least one variable, then these variables are *not* read-only.

The concept of read-only variables provides CP with a powerful programming technique, called partially determined messages.

Now we know what a CP-program looks like and how the unification algorithm works, we will give an informal description of the CP interpreter, which will model our operational semantics.

THE CP INTERPRETER

The execution of an atom is called a proces. A set of processes is called a system. The execution of a CP system S running a program P can be described informally as follows:

Each process $A1$ in S tries to unify $A1$ with each head of a clause $A \leftarrow G : B$ in P in parallel. If the unification is successful the guard system G is executed. As soon as one of the guard systems terminates successfully and can commit $A1$ is replaced by the system B and all the other guard systems are discarded. The system S terminates when it is empty. It may become empty only if some of the clauses in P have empty bodies.

The computation of a CP-program causes a hierarchy of systems. Each process may invoke several guard systems with the intention to find a reducing clause and each of these guard systems may create other systems as well. The communication between these systems is governed by the commitment

mechanism. Each system created by $A1$ has only access to variables which occur in $A1$, which are private copies of the global ones. When a commitment is made, the variables are unified against their global counter parts and if the unification succeeds the body system B of the chosen clause replaces A.

We will now give a more detailed description of the CP interpreter. The CP interpreter uses three kinds of processes: an *and-dispatcher,* an *or-dispatcher* and a *unifier.* These processes should not be confused with the CP processes themselves, which are unit goals.
We begin with a system S of a CP process. After an and-dispatcher is invoked with S, the computation proceeds as follows:

- An *and-dispatcher,* invoked with system S, spawns an or-dispatcher for every CP process in S, and waits for all its child or-dispatchers to report success. When they do, it reports success and terminates.
- An *or-dispatcher,* invoked with a CP process A, operates as follows. For every clause $A1 \leftarrow G : B$, whose head is potentially unifiable with A, it invokes a unifier with A and the clause $A1 \leftarrow G : B$. Following that the or-dispatcher waits for any of the unifiers to report success. When *one* such report arrives, the or-dispatcher reports success to its parent and-dispatcher and terminates.
- A *unifier,* invoked with a CP process A and a guarded clause $A1 \leftarrow G : B$, operates as follows:
  It *attempts* to unify A with $A1$, storing bindings made to non read-only variables in A on a private storage. If and when successful, it invokes an and dispatcher with G and waits for it to report success.
  When this report arrives, the unifier attempts to commit, as explained below. If the commitment completed successfully it reports success, but in either case it terminates.

At most one unifier spawned by an or-dispatcher may commit. This mutual exclusion can be achieved by the standard techniques developed for this kind of problems. When a unifier wants to commit, it first has to have permission to do so. If it has, it will try to unify the local copies of the variables with the corresponding global ones and if successful, then the commitment completes successfully.

Some remarks: In the description of the unifier process, we wrote *attempts to unify,* because it is possible that a unification fails due to read-only constraints, in that case unification may succeed later. In this case we will call the process A with the clause $A1 \leftarrow G : B$ a suspended process. The process of unification is therefore a continuous activity, which terminates only on success and when failure is not caused by violation of the read-only constraints, because in the latter case it can not be cured in the future.

THE CP OPERATIONAL MODEL

The following part of this chapter is a description of the operational model of CP. The three processes described in the previous section build a hierarchical structure of systems. Systems contain unit goals and/or other systems. These systems will be expressed in our model by some new syntactic operators.
Because all the atoms in a conjunction are developed in parallel, we must take care of renaming of variables. The following may not happen: suppose that during the reduction of a_1 a variable occurs which was not in a_1 and a_2, that this variable occurs also in the reduction of a_2 and that the values of this variable in the two processes differ. Possibly one unification against the global counterpart succeeds, while the other fails. The second unification would have been possible if the variable was properly renamed.
To overcome this problem we introduce a countably number of disjoint sets with countable many variables:

(3.28) DEFINITION Let μ and τ be elements of NUM* then:
$VAR_\epsilon = \{ x_1, x_2, x_3, .. \}$
$VAR_\tau = \{ x_{1,\tau}, x_{2,\tau}, x_{3,\tau}, .. \}$

$VAR = \cup \{ VAR_\tau \mid \tau \in NUM^* \}$.
We fix renamings $\psi_\mu : VAR_\epsilon \rightarrow VAR_\mu$, with $\psi_\mu(x_j) = x_{j,\mu}$

In our model a process A consists of an atom with renaming index, a substitution (the local copies of the variables), a sentence and a renaming index for that sentence and some local status information. All the processes on the same level have a joint global status, which contains information about the global variables, the number of processes currently running on this level and a state of activeness. A process can also be a system. This occurs when the guard system is set up by the unifier. The local status information tells us if we are in an active, suspended or an update state and if we develop a guard or a body or the process wants to commit. The global status denotes the activity of the system, this can be just active or in a update state. A process is in a suspended state if it tries to unify but due to read-only constraints fails. Then it has to wait for the information to be delivered when its local variables are updated (this occurs when the global status is in the update state).
If a process does commit, then the system goes in the update state, denoted by the global status. In this state all the local copies of the variables of the different processes are updated with the corresponding global ones, if possible.

A BRIEF DESCRIPTION OF OUR MODEL

In our model we start with a goal, say $a_1 \wedge .. \wedge a_n$. For each a_i a process is made (compare this with the and-dispatcher of Shapiro's interpreter). All these processes form a system.
Then for all the clauses in P a process is set up with a_i (compare this with the or-dispatcher)
Then if the head of the clause is unifiable with a_i, processes are created to solve the guards. If one of the guard systems of an atom a_i is finished successfully and commits, then all the other guard systems of the atom a_i are discarded and the body is added to the system we began with (compare this with the unifier process).

THE OPERATIONAL MODEL

First we have some definitions. We introduce some new classes: *GBODY*, *STATE*, *BODY_STATE*, *GLOBAL_STATE*, *AND_STATE*, *OR_STATE*, *AS* and *GB* respectively with typical elements: gb, σ, σ_b, σ_{gl}, σ_{AND}, σ_{OR}, T and $\bar{V}$ respectively.

(3.29) DEFINITION
$STATE = BODY\_STATE \times NUM^* \times SUBST \times PROG \times NUM^* \times AS \times GB$
$GLOBAL\_STATE = AND\_STATE \times (AS \times NUM \times NUM \times SUBST)$
Let $c \in CONJ$ then $c ::= a \mid c_1 \wedge c_2$
Let $\sigma_{OR} \in OR\_STATE$ then $\sigma_{OR} ::= \sigma \mid \sigma_{OR_1} \parallel \sigma_{OR_2}$
Let $\sigma_{AND} \in AND\_STATE$ then $\sigma_{AND} ::= \sigma_{OR} \mid \sigma_{AND_1} \square \sigma_{AND_2}$
Let $\sigma_b \in BODY\_STATE$ then $\sigma_b ::= gb \mid \sigma_{gl} : b$
Let $gb \in GBODY$ then $gb ::= b \mid b_1 : b_2$
The set *AS* has as elements *A*, *S* and *U*, which tell us if the process is in a *A*ctive- *S*uspended- or *U*pdate state.
The set *GB* has as its elements *G*, *B* and *C*, which tell us if a process is part of a *B*ody or *G*uard of a clause or it shows us it wants to make a *C*ommitment.
For the classes *BODY*, *SUBST*, *NUM* and *PROG* we use definitions as defined in chapter 2. In the previous chapters we had a goal b with $index(b) \leqslant m$. This meant that $var(b) \subseteq \cup_{i=0}^{m} VAR_i$, but since we have a more complex partition of *VAR*, we cannot do this any more. From now on for all $b \in BODY$, $var(b) \subseteq VAR_\epsilon$ and in our model (b, μ) stand for $\psi_\mu(b)$.
The transition relation "$\rightarrow$" is defined on *GLOBAL_STATE* to *GLOBAL_STATE* $\cup$ (*AS* $\times$ *NUM* $\times$ *NUM* $\times$ *SUBST*) $\cup \{ \epsilon \}$
We define for the sentence $P \leftarrow b$ the start goal by: $<[b, \epsilon, Id, P, 0, A, B], (A, 1, 0, Id)>$.

In the following definition T stands for a A or S (not U) and V stands for G or B (not C).
Let $|c|$ stands for the number of atoms in c, so $|a_0 \wedge a_1 \wedge a_2| = 3$.

THE TRANSITION RULES

AND-parallelism

$$< [c_1 \wedge c_2,\ \mu,\ \phi,\ P',\ \tau \bullet q,\ A,\ V],\ (T,\ n,\ 0,\ \phi') > \tag{1}$$

$$\rightarrow < [c_1,\ \mu,\ \phi,\ P',\ \tau \bullet q,\ A,\ V] \square [c_2,\ \mu,\ \phi,\ P',\ \tau \bullet (q + |c_1|),\ A,\ V],\ (T,\ n+1,\ 0,\ \phi') >$$

OR-parallelism

$$< [a,\ \mu,\ \phi,\ P_1 \| P_2,\ \tau,\ A,\ V],\ (T,\ n,\ 0,\ \phi') > \tag{2}$$

$$\rightarrow < [a,\ \mu,\ \phi,\ P_1,\ \tau,\ A,\ V] \| [a,\ \mu,\ \phi,\ P_1,\ \tau,\ A,\ V],\ (T,\ n+1,\ 0,\ \phi') >$$

Guard evaluation

$$< [b_1 : b_2,\ \mu,\ \phi,\ P',\ \tau \bullet q,\ A,\ V],\ (T,\ n,\ 0,\ \phi') > \tag{3}$$

$$\rightarrow < [\ < [b_1,\ \mu,\ \phi,\ P',\ \tau \bullet q,\ A,\ V],\ (A,\ 1,\ 0,\ \phi) > : b_2,\ \mu,\ \phi,\ P,\ \tau \bullet (q + |c_1|),\ A,\ \mathbf{G}],$$
$$(T,\ n,\ 0,\ \phi') >$$

Unifier process

$$< [a,\ \mu,\ \phi,\ a' \leftarrow b_1' : b_2',\ \tau,\ A,\ V],\ (T,\ n,\ 0,\ \phi') > \tag{4a}$$

$$\rightarrow < [b_1' : b_2',\ \tau \bullet 0,\ (\theta \circ \phi),\ P,\ \tau \bullet 0,\ A,\ \mathbf{G}],\ (T,\ n,\ 0,\ \phi') >$$

if $\theta = mgu(\phi(\psi_\mu(a)), \psi_{\tau \bullet 0}(a'))$ exists

$$\rightarrow < [a,\ \mu,\ \phi,\ a' \leftarrow b_1' : b_2',\ \tau,\ \mathbf{S},\ V],\ (T,\ n,\ 0,\ \phi') >$$

if $\theta = mgu(\phi(\psi_\mu(a)), \psi_{\tau \bullet 0}(a'))$ suspends

$$\rightarrow \epsilon \text{ otherwise}$$

$$< [a,\ \mu,\ \phi,\ a' \leftarrow b',\ \tau,\ A,\ V],\ (T,\ n,\ 0,\ \phi') > \tag{4b}$$

$$\rightarrow < [b',\ \tau \bullet 0,\ (\theta \circ \phi),\ P,\ \tau \bullet 0,\ A,\ \mathbf{C}],\ (T,\ n,\ 0,\ \phi') >$$

if $\theta = mgu(\phi(\psi_\mu(a)), \psi_{\tau \bullet 0}(a'))$ exists

$$\rightarrow < [a,\ \mu,\ \phi,\ a' \leftarrow b',\ \tau,\ \mathbf{S},\ V],\ (T,\ n,\ 0,\ \phi') >$$

if $\theta = mgu(\phi(\psi_\mu(a)), \psi_{\tau \bullet 0}(a'))$ suspends

$$\rightarrow \epsilon \text{ otherwise}$$

Evaluate guard

$$< [\ < (T_1,\ n_1,\ 0,\ \phi_1) > : b,\ \mu,\ \phi,\ P',\ \tau,\ A,\ V],\ (T_2,\ n_2,\ 0,\ \phi_2) > \tag{5}$$

$$\rightarrow < [b,\ \mu,\ (\phi_1 \circ \phi),\ P,\ \tau,\ A,\ \mathbf{C}],\ (T_2,\ n_2,\ 0,\ \phi_2) >$$

Committing

$$< [b,\ \mu,\ \phi_1,\ P,\ \tau\ A,\ \mathbf{C}],\ (T,\ n,\ 0,\ \phi_2) > \tag{6}$$

$\rightarrow < [b,\ \mu,\ (\theta \circ \phi_1),\ P,\ \tau\ \mathbf{U},\ \mathbf{B}],\ (\mathbf{U},\ n,\ 1,\ (\theta \circ \phi_2)) >$

if $\theta = mgu(\phi_1, \phi_2)$ exists

$\rightarrow \epsilon$ otherwise

Going in Update state

$$< [\mathbf{true},\ \mu,\ \phi,\ P',\ \tau,\ A,\ V],\ (T,\ n,\ 0,\ \phi') > \rightarrow < (\mathbf{U},\ n-1,\ 0,\ (\theta \circ \phi)) > \quad (7)$$

if $\theta = mgu(\phi, \phi')$ exists

$\rightarrow \epsilon$ otherwise

Transition in update state

$$< [gb,\ \mu,\ \phi_1,\ P',\ \tau\ T,\ V],\ (\mathbf{U},\ n,\ p,\ \phi_2) > \quad (8)$$

$\rightarrow < [b,\ \mu,\ (\theta \circ \phi_1),\ P',\ \tau\ \mathbf{U},\ V],\ (\mathbf{U},\ n,\ p+1,\ \phi_2)) >$

if $\theta = mgu(\phi_1, \phi_2)$ exists

$\rightarrow \epsilon$ otherwise

$$< [\ <\sigma_{AND},\ (T,\ n,\ 0,\ \phi_1) > : b,\ \mu,\ \phi_2,\ P',\ \tau\ A,\ V],\ (\mathbf{U},\ n',\ p,\ \phi_3) > \quad (9)$$

$$\rightarrow < [\ <\sigma_{AND},\ (\mathbf{U},\ n,\ 0,\ (\theta \circ \phi_1)) > : b,\ \mu,\ \phi_2,\ P',\ \tau\ \mathbf{U},\ V],\ (\mathbf{U},\ n,\ p+1,\ \phi_3)) >$$

if $\theta = mgu(\phi_1, \phi_3)$ exists

$\rightarrow \epsilon$ otherwise

From update state to active state

$$< \sigma_{AND},\ (\mathbf{U},\ n,\ n,\ \phi) > \rightarrow < \sigma_{AND},\ (\mathbf{A},\ n,\ n,\ \phi) > \quad (10)$$

$$< [\sigma_b,\ \mu,\ \phi,\ P',\ \tau,\ \mathbf{U},\ V],\ (A,\ n,\ p,\ \phi') > \quad (11)$$

$$\rightarrow < [\sigma_b,\ \mu,\ \phi,\ P',\ \tau,\ \mathbf{A},\ V],\ (A,\ n,\ p-1,\ \phi') >$$

Now we have defined the transition rules, we have to define some inference rules also. The T, T' and T'' can now have the values A, S and U.

THE INFERENCE RULES

OR-parallelism

$$\frac{< \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < [\sigma,\ B\ ],\ (T',\ n',\ p',\ \phi') >}{< \sigma_{OR_1} \| \sigma_{OR},(T,\ n,\ p,\ \phi) > \rightarrow < [\sigma,\ B\ ],\ (T',\ n'-1,\ p',\ \phi') >} \quad (12a)$$

$$\frac{< \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < [\sigma,\ B\ ],\ (T',\ n',\ p',\ \phi') >}{< \sigma_{OR} \| \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < [\sigma,\ B\ ],\ (T',\ n'-1,\ p',\ \phi') >} \quad (12b)$$

$$\frac{< \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{OR_2},\ (T',\ n',\ p',\ \phi') >}{< \sigma_{OR_1} \| \sigma_{OR},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{OR_2} \| \sigma_{OR},\ (T',\ n',\ p',\ \phi') >} \quad (12c)$$

$$\frac{< \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{OR_2},\ (T',\ n',\ p',\ \phi') >}{< \sigma_{OR} \| \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{OR} \| \sigma_{OR_2},\ (T',\ n',\ p',\ \phi') >} \quad (12d)$$

$$\frac{< \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon}{< \sigma_{OR_1} \| \sigma_{OR},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{OR},\ (T,\ n-1,\ p,\ \phi) >} \quad (12e)$$

$$\frac{< \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon}{< \sigma_{OR} \| \sigma_{OR_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{OR},\ (T,\ n-1,\ p,\ \phi) >} \quad (12f)$$

AND-parallelism

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND_2},\ (T',\ n',\ p',\ \phi') >}{< \sigma_{AND_1} \square \sigma_{AND},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND_2} \square \sigma_{AND},\ (T',\ n',\ p',\ \phi') >} \quad (13a)$$

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND_2},\ (T',\ n',\ p',\ \phi') >}{< \sigma_{AND} \square \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND} \square \sigma_{AND_2},\ (T',\ n',\ p',\ \phi') >} \quad (13b)$$

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < (U',\ n',\ p',\ \phi') >}{< \sigma_{AND_1} \square \sigma_{AND},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND},\ (U',\ n',\ p',\ \phi') >} \quad (13c)$$

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < (U',\ n',\ p',\ \phi') >}{< \sigma_{AND} \square \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND},\ (U',\ n',\ p',\ \phi') >} \quad (13d)$$

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon}{< \sigma_{AND_1} \square \sigma_{AND},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon} \quad (13e)$$

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon}{< \sigma_{AND} \square \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon} \quad (13f)$$

Guard evaluation

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow < \sigma_{AND_2},\ (T',\ n',\ p',\ \phi') >}{< [\ <\sigma_{AND_1},\ (T,\ n,\ p,\ \phi) > : b,\ \mu,\ \phi_2,\ P',\ \tau,\ T_1,\ V],\ (T'',\ n'',\ p'',\ \phi'') > \rightarrow < [\ <\sigma_{AND_2},\ (T',\ n',\ p',\ \phi') > : b,\ \mu,\ \phi_2,\ P',\ \tau,\ T_1,\ V],\ (T'',\ n'',\ p'',\ \phi'')) >} \quad (14a)$$

$$\frac{< \sigma_{AND_1},(T,\ n,\ p,\ \phi) > \rightarrow \epsilon}{< [<\sigma_{AND_1},\ (T,\ n,\ p,\ \phi) > : b,\ \mu,\ \phi_2,\ P',\ \tau\ T,\ V],\ (T'',\ n'',\ p'',\ \phi'') > \rightarrow \epsilon} \quad (14b)$$

Remarks: Transition rules (1) and (2) establish the so called AND- and OR-parallelism. Notice that rule (2) is alike rule (5) of the definition of the operational semantics in the first part of this chapter. Rule (3) together with inference rule (14) shows us that to evaluate a guarded-body, we must first evaluate the guard. In rule (4b) we have a clause with an empty guard, so if the mgu exists this process is ready to commit. If a guard has been successfully evaluated (rule (5)) then the process wants to commit (note: ϕ_1 can be written as $\phi_1' \circ \phi$). A process can commit if the mgu of the two substitutions exists. † If a commitment has been made the system is in the update state. This happens also if

† The mgu of two substitution with finite domain, ϕ_1 and ϕ_2 can be calculated as follows:
Let $dom(\phi_1) \cup dom(\phi_2) = \{ y_1,..,y_n \}$. Let p be a new predicate symbol of arity n then
$mgu(\ \phi_1\ ,\ \phi_2\ ) ::= mgu(\ \phi_1(p(y_1,..,y_n))\ ,\ \phi_2(p(y_1,..,y_n))\ )$

a guarded-body has been successful (rule (7)). All the processes are suspended until they have received the global information (rules (8) - (12)). In inference rules (12a) and (12b) $[\sigma,B]$ is an element of *STATE*. If a σ_{OR} evaluates to $[\sigma,B]$, this means the commitment has been successful, then all the other processes, which were in "OR-parallel" with the first one are discarded (rules (12a) and (12b)).

We have four kinds of production sequences:

(1) σ_{gl_0}, σ_{gl_1}, σ_{gl_2},..,$(A, 0, 0, \phi)$

(2) σ_{gl_0}, σ_{gl_1}, σ_{gl_2},..., σ_{gl_n}

(3) σ_{gl_0}, σ_{gl_1}, σ_{gl_2},..., ϵ

(4) σ_{gl_0}, σ_{gl_1}, σ_{gl_2},.. ..

where $\sigma_{gl_n} \rightarrow \sigma_{gl_{n+1}}$ holds for $n = 0, 1, 2,...$ $\perp$ is defined as before. In possibility (2) σ_{gl_n} is not equal to ϵ or (T, n, p, ϕ). Production sequence (1) is the successful one. The second (2) occurs if all the processes are suspended and there is not a process running which can activate the system. (3) happens when the goal is not satisfiable. (4) if the computation never ends.

We say that $\sigma_{gl} \rightarrow^{\infty} W$, where $W \in \mathcal{P}(\ SUBST \cup \{ \perp \}\ )$, if one of the following conditions hold:

1. In case production sequence 1. then $W = \{ \phi \}$
2. In case 2. and 3. then $W = \varnothing$
3. In case 4. $W = \{ \perp \}$.

The operational semantics can be defined as:

$\mathcal{O}(\ P \leftarrow b\ ) = \cup \{\ W \mid <[b, \epsilon, Id, P, 0, A, B], (A, 1, 0, Id)> \rightarrow^{\infty} W\ \}$.

With this last definition we have reached the end of this chapter. We do not give a denotational semantics of CP because we do not have one. Note that $\mathcal{O}(P \leftarrow b) \in \mathcal{P}(\ SUBST \cup \{ \perp \})$. Moreover it is probably the case that if $\mathcal{O}(P \leftarrow b)$ is infinite, $\perp$ is an element of $\mathcal{O}(P \leftarrow b)$, but we do not know it for sure. We do not know enough of the effects of the read-only variables and the commitment operator on the production sequences. In the epilog we will say something more about further research areas.

Chapter 4

Infinite computations

In chapter 2 we have been interested in the result of a computation only in case the derivation was finite. However, there are cases where the computation is infinite, but the generated sequence of most general unifiers still makes sense.

Consider for example the sequence of Fibonacci numbers $(F_n)_{n=0}^{\infty}$:

$$F_0 = F_1 = 1 \ , \qquad F_{n+2} = F_{n+1} + F_n \ .$$

We can write this sequence as an infinite list, such that the n-th atom of the list is the n-th Fibonacci-number:

$$F = 1.1.2.3.5.8.13. \cdots$$

For the list F we have the following fixpoint equation:

$$0.0.F + 1.F = F \ .$$

With this in mind we can write a logic program that computes *F*.
First we formalize addition of infinite lists by the following Horn clause:

(i): lsum (a.x,b.y,c.z) $\leftarrow$ sum (a,b,c) $\wedge$ lsum (x,y,z).

where "lsum (x,y,z)" means "the (finite or infinite) list z is the sum of the (finite or infinite) lists x and y", and "sum (a,b,c)" means "a + b = c". The fixpoint equation for the list of Fibonacci-numbers can be formalized by the Horn clause:

(ii): $\leftarrow$ lsum (0.0.f,1.f,f).

Clause (i) and (ii), together with an appropriate definition of sum yields an infinite computation which generates successive approximations of the list of Fibonacci-numbers:

$1.f_1$

$1.1.f_2$

$1.1.2.f_3$

etc.

Every Fibonacci number will eventually be produced by the computation.

In the sequel of this chapter we fix some finite logic program P. Let A be the alphabet of P and C, F resp. R the collection of constants, function symbols and relation symbols of A. Moreover we adopt the definitions of chapter 2.1.

Let HU be the collection of all terms over the alphabet A. Let Ω be a constant not in A. Define HU' as the collection of all terms over the alphabet $A \cup \{\Omega\}$. and the cut $\alpha_n: HU \to HU'$ at height n, $n \geqslant 0$ inductively as follows:

$$\alpha_0(t) = \Omega \quad \text{for } \forall\, t \in HU$$

$$\alpha_{n+1}(t) = \begin{cases} t & \text{if } t \in C \\ f(\alpha_n(t_1),..,\alpha_n(t_k)) & \text{if } t = f(t_1,..,t_k) \end{cases}$$

We turn HU into a metric space with distance d where

$$d(t,t') = \begin{cases} 0 & \text{if } t = t' \\ 2^{-m} & \text{if } m = inf\{\ n \mid \alpha_n(t) \neq \alpha_n(t')\} \end{cases}$$

The completed Herbrand universe $\overline{HU}$ is defined as the completion of HU with respect to d. We can consider $\overline{HU}$ as the collection of all finite and infinite terms over the alphabet $A \cup \{\Omega\}$. Similarly the completed Herbrand base $\overline{HB}$ is defined as the completion of HB with respect to the distance d, where for $a = R(t_1,..,t_k)$, $a' = R'(t'_1,..,t'_l) \in HB$:

$$d(a,a') = \begin{cases} 1 & \text{if } R \neq R' \\ \tfrac{1}{2} \min\{\ d(t_i,t'_i) \mid 1 \leqslant i \leqslant k\ \} & \text{if } R = R' \end{cases}$$

Note that $\overline{HU}$ and $\overline{HB}$ are compact, because P is finite.
A continuous Herbrand interpretation is a closed subset of $\overline{HB}$. A continuous Herbrand interpretation I induces an interpreting structure $(\overline{HU},*)$, where
1. if $c \in C$ then $c^* = c$;
2. if $f \in F$ k-ary then $f^*(t_1,..,t_k) = f(t_1,..,t_k)$;
3. if $r \in R$ k-ary then $r^* = \{\ (t_1,..,t_k) \mid r(t_1,..,t_k) \in I\ \}$
When $\{ff, tt\}$ is equipped with the topology $\{\ \varnothing, \{ff\}, \{ff,tt\}\ \}$, then the map which assigns to a (finite or infinite) atom its truth value in $(\overline{HU},.^*)$ is continuous.

An infinite derivation Δ from c is an infinite sequence $(g_i)_{i=0}^{\infty}$ such that
1. $g_0 = <c,id,m>$ where $m = index(c)$;
2. $g_{i-1} \to g_i$, $i \geqslant 1$.
If $\Delta = (g_i)_{i=0}^{\infty}$ is an infinite derivation then $\Delta \mid_k$ is the finite derivation $\Delta \mid_k = (g_i)_{i=0}^{k}$. $\Delta \mid_k$ is the initial segment of Δ of length k. The set $[\Delta(a)]$ computed by the infinite derivation Δ from an atom a is defined by $[\Delta(a)] = \cap_{k=0}^{\infty} [\Delta \mid_k (a)]$. Since $\overline{HB}$ is compact this set is never empty.

(4.1) DEFINITION Let $\Delta = (g_k)_{k \in I}$ be a derivation. We define the function δ' from the atoms of the goals g_k, $k \in I$ to the integers and the function δ from $\{g_k \mid k \in I\}$ to the integers. If $g_0 = (\wedge_{i=1}^{s} a_i, id, m)$ then we define $\delta'(a_i) = 0$, $1 \leqslant i \leqslant s$. If $g_k \to g_{k+1}$ by resolution of the atom a_j, $j \in \{1,..,s\}$ in g_k against $a_0' \leftarrow a_1' \wedge .. \wedge a_r'$ then we define $\delta'(a_i') = \delta'(a_j) + 1$, $1 \leqslant i \leqslant r$. If $g_k = (\wedge_{i=1}^{s} a_i, \phi, n)$ then we define $\delta(g_k) = \min\{\ \delta'(a_i) \mid 1 \leqslant i \leqslant s\}$ with the convention $\min(\varnothing) = \infty$.
We call Δ fair up to q, $(q \geqslant 0)$ if $\delta(g_k) = q$ for some $k \in I$. Note that every successful derivation is a fair one.

(4.2) DEFINITION The collection $\mathcal{P}(\overline{HB})$ of closed subsets of $\overline{HB}$ is defined by $\mathcal{P}(\overline{HB}) = \{\ I \subseteq \overline{HB} \mid I \text{ closed}\ \}$. The transformation $\mathcal{T}$ associated with the logic program P is defined by $\mathcal{T}(I) = \{\ a \in \overline{HB} \mid a_0' \leftarrow a_1' \wedge .. \wedge a_s' \in P;\ \sigma \in SUBST;\ \sigma(a_0') = a;\ \sigma(a_1'), .. , \sigma(a_s') \in I\ \}$.

(4.3) PROPOSITION $\mathcal{T}: \mathcal{P}(\overline{HB}) \to \mathcal{P}(\overline{HB})$ is well defined.

PROOF Let $I \in \mathcal{P}(\overline{HB})$ and $(a_i)_{i=0}^{\infty}$ a Cauchy sequence in $\mathcal{T}(I)$. Since P is finite we can assume without loss of generality the existence of a clause $a_0' \leftarrow a_1 \wedge .. \wedge a_s'$ in P and of a sequence $(\sigma_i)_{i=0}^{\infty}$ in $SUBST$ such that $dom(\sigma_i) \subseteq var(a_0',..,a_s')$, $a_i = \sigma_i(a_0')$ and $\sigma_i(a_1'),.., \sigma_i(a_s') \in I$, $i \geqslant 0$.
By compactness of $\overline{HB}$ there exist a subsequence $(\sigma_{i_k})_{k=0}^{\infty}$ and $\sigma \in SUBST$ such that $\sigma_{i_k}(x) \to \sigma(x)$

and for each $x \in var(a_0',..,a_s')$. Put $a=\sigma(a_0')$. Then we have $\sigma(a_i') \in I$, $1 \leqslant i \leqslant s$, because I is closed. Thus $a \in \mathcal{T}(I)$. Moreover $a_i \to a$, since $\sigma_{i_k}(a_0') \to \sigma(a_0')$. □

(4.4) PROPOSITION (i). $\mathcal{P}(\overline{HB})$ ordered by inclusion is a complete lattice.
(ii). $\mathcal{T}: \mathcal{P}(\overline{HB}) \to \mathcal{P}(\overline{HB})$ is monotonic.

PROOF (i). If $\{ I_\xi \mid \xi \in \Xi \}$ is a subset of $\mathcal{P}(\overline{HB})$, take $\overline{\cup_{\xi \in \Xi} I_\xi}$ as its least upper bound.
(ii). Straightforward from the definition of $\mathcal{T}$. □

(4.5) PROPOSITION $\mathcal{P}(\overline{HB})$ ordered by reversed inclusion is a complete lattice.

PROOF Take $\overline{HB}$ as bottom and if $\{ I_\xi \mid \xi \in \Xi \}$. is a subset of $\mathcal{P}(\overline{HB})$, take the closed subset $\cap_{\xi \in \Xi} I_\xi$ of $\overline{HB}$ as the least upperbound. □

(4.6) PROPOSITION $\mathcal{T}: \mathcal{P}(\overline{HB}) \to \mathcal{P}(\overline{HB})$ is continuous (with respect to reversed inclusion).

PROOF The monotonicity of $\mathcal{T}$ is again straightforward from the definition. Hence it is sufficient to prove: for each chain $(I_n)_{n=0}^{\infty}$ in $\mathcal{P}(\overline{HB})$ we have $\cap_{n=0}^{\infty} \mathcal{T}(I_n) \subseteq \mathcal{T}(\cap_{n=o}^{\infty} I_n)$.
So let $(I_n)_{n=0}^{\infty}$ in $\mathcal{P}(\overline{HB})$ and $x \in \cap_{n=0}^{\infty} \mathcal{T}(I_n)$. For each $n \geqslant 0$ there exists a clause $\alpha^{(n)} \in P$ such that $x \in \mathcal{T}(I_n)$ via $\alpha^{(n)}$. Since P is finite and $(I_n)_{n=0}^{\infty}$ is a chain we can assume without loss of generality that there exists a clause $\alpha \in P$ such that $x \in \mathcal{T}(I_n)$ via α, for all $n \geqslant 0$. Say $\alpha=(a_0 \leftarrow a_1 \wedge .. \wedge a_s)$. Hence for each $n \geqslant 0$ there exists $\sigma_n \in SUBST$ such that $dom(\sigma_n) \subseteq var(a_0,..,a_n)$, $x=\sigma_n(a_0)$ and $\sigma_n(a_1),.., \sigma_n(a_s) \in I_n$.
By compactness of $\overline{HB}$ there exists a convergent subsequence $(\sigma_{n_k})_{k=0}^{\infty}$ of $(\sigma_n)_{n=0}^{\infty}$. Let σ denote the limit of $(\sigma_{n_k})_{k=0}^{\infty}$. Clearly we have $\sigma(a_0)=x$. Since $\sigma_{n_k}(a_i) \to \sigma(a_i)$, $k \to \infty$ for $1 \leqslant i \leqslant s$ we have $\sigma(a_i) \in \cap_{k=0}^{\infty} I_{n_k} = \cap_{n=0}^{\infty} I_n$, $1 \leqslant i \leqslant s$. But then $x \in \mathcal{T}(\cap_{n=0}^{\infty} I_n)$. □

(4.7) COROLLARY $\cap_{n=0}^{\infty} \mathcal{T}^n(\overline{HB})$ is the greatest fixpoint of $\mathcal{T}$, (with respect to inclusion).

PROOF In general we have $\cap_{n=0}^{\infty} \mathcal{T}^n(\overline{HB}) \supseteq gfp(\mathcal{T})$; by the above proposition $\cap_{n=0}^{\infty} \mathcal{T}^n(\overline{HB})$ is a fixpoint of $\mathcal{T}$. Hence $\cap_{n=0}^{\infty} \mathcal{T}^n(\overline{HB})$ is the greatest fixpoint of $\mathcal{T}$. □

(4.8) LEMMA Let a be an atom and $\Delta=(g_i)_{i=0}^{k}$ a derivation from a and fair up to q. Then $[\Delta(a)] \subseteq \mathcal{T}^q(\overline{HB})$.

PROOF By induction on q. Case $q=0$: trivial.
Case $q+1 \leftarrow q$: Let $\Delta=(g_i)_{i=0}^{k}$ be a derivation from a, fair up to $q+1$ with product of unifiers ϕ. Let $a_0' \leftarrow a_1' \wedge .. \wedge a_s' \in P_1$ and $\theta \in SUBST$ such that $g_0 \to g_1$ via $a_0' \leftarrow a_1' \wedge .. \wedge a_s'$ and mgu θ.
If $s=0$, then clearly $[\Delta(a)] \subseteq \mathcal{T}^{q+1}(\overline{HB})$, because $\theta(a)=\theta(a_0')$ and $\phi=\theta$. So assume $s \geqslant 1$. Because there exists a finite derivation from $\theta(a_j')$, fair up to q with product of unifiers ϕ_j, we conclude from the induction hypothesis that $[\phi_j(\theta(a_j'))] \subseteq \mathcal{T}^q(\overline{HB})$, $1 \leqslant j \leqslant s$. Thus $[\phi(a_j')] \subseteq \mathcal{T}^q(\overline{HB})$, since $[\phi(a_j')] \subseteq [\phi_j(\theta(a_j'))]$, $1 \leqslant j \leqslant s$. So $[\Delta(a)] = [\phi(a)] = [\phi(a_0')] \subseteq \mathcal{T}^{q+1}(\overline{HB})$. □

(4.9) LEMMA Let a be an atom. Then:
$[a] \cap \mathcal{T}^q(\overline{HB}) \subseteq \cup \{ [\Delta(a)] \mid \Delta \text{ finite and fair up to q} \}$

PROOF By induction on q. Case $q=0$: trivial.
Case $q+1 \leftarrow q$: Let a be an atom and $a' \in [a] \cap \mathcal{T}^{q+1}(\overline{HB})$. Let $\sigma_1 \in SUBST$ such that $\sigma_1(a)=a'$. Let $a_0 \leftarrow a_1 \wedge .. \wedge a_s \in P$ and $\sigma_2 \in SUBST$ such that $\sigma_2(a_0)=a'$ and $\sigma_2(a_1),.., \sigma_2(a_s) \in \mathcal{T}^q(\overline{HB})$. Without loss of generality $s \geqslant 1$. Assume a is of index m. Let $a_0' \leftarrow a_1' \wedge .. \wedge a_s'$ the variant of $a_0 \leftarrow a_1 \wedge .. \wedge a_s$ in P_{m+1}. Let $\sigma \in SUBST$ such that $\sigma(a)=a'$, $\sigma(a_0')=a'$ and $\sigma(a_1'),..,\sigma(a_s') \in \mathcal{T}^q(\overline{HB})$.
By the induction hypothesis, there exists a finite derivation from $\sigma(a_i')$ fair up to q, $1 \leqslant i \leqslant s$. So there exists a finite derivation from $\wedge_1^s \sigma(a_i')$ fair up to q. Let $\theta=mgu(\sigma(a),a_0')$, which exists since $\sigma(\sigma(a))=\sigma(a_0')$. By the rectangle lemma there also exists a finite derivation from $\wedge_1^s \theta(a_i')$ fair up to q. Hence there exists a finite derivation Δ from $\sigma(a)$ fair up to $q+1$. Again by the rectangle lemma, there exists a finite derivation Δ' from a, fair up to $q+1$ such that $[\Delta(\sigma(a))] \subseteq [\Delta'(a)]$. Then $a'=\sigma(a)$

$\in \{\sigma(a)\} = [\Delta(\sigma(a))] \subseteq [\Delta'(a)]$ and Δ' is finite and fair up to q+1. □

(4.10) THEOREM Let a be an atom. Then: $[a] \cap gfp(\mathfrak{T}) = \cup\{ [\Delta(a)] \mid \Delta \text{ fair from } a\}$.

PROOF "$\subseteq$": Let $a' \in [a] \cap gfp(\mathfrak{T})$. Then $a' \in [a] \cap \mathfrak{T}^q(\overline{HB})$, so by lemma (4.9) there exists a finite derivation Δ_q from a, fair up to q such that $a' \in [\Delta(a)]$, for each $q \geq 0$. Note that without loss of generality Δ_q can be assumed to be leftmost.
Define E_q, $q \geq 0$ as the collection of all derivations Δ from a, exactly fair up to q, with left-right computation rule such that $a' \in [\Delta(a)]$. Every E_q is finite and non-empty, $q \geq 0$ and for each element of E_{q+1} there exists an element of E_q, which is an initial segment. By König's lemma there exists a infinite derivation Δ from a, which is fair. Moreover $a' \in \cap_{q=0}^{\infty} [\Delta_q(a)] \subseteq [\Delta(a)]$.
"$\supseteq$". Let $a' \in [\Delta(a)]$, where Δ is a fair derivation from a. Let Δ_q be an initial segment of Δ that is fair up to q, $q \geq 0$. Then $a' \in [\Delta_q(a)]$ because $[\Delta(a)] \subseteq [\Delta_q(a)]$, $q \geq 0$. So $a' \in [a] \cap \mathfrak{T}^q(\overline{HB})$ by lemma (4.8), for each $q \geq 0$. $a' \in [a] \cap \cap_{q=0}^{\infty} \mathfrak{T}^q(\overline{HB})$. Hence $[\Delta(a)] \subseteq [a] \cap gfp(\mathfrak{T})$. □

(4.11) DEFINITION Let a be an infinite atom. We call a computable at infinity, if there exist a *finite* atom a' and a fair derivation Δ such that $\phi_n(a') \rightarrow a$, $k \rightarrow \infty$, where ϕ_n is the result of $\Delta \mid_n$.

In the above situation we have $[\Delta(a')] = \{a\}$. Moreover if $[\Delta(a')] = \{a\}$ for some Δ and finite atom a', then a is computable at infinity.

(4.12) THEOREM (i): Let a be an infinite atom. If a is computable at infinity, then $a \in gfp(\mathfrak{T})$. (ii): For the program $p(f(x)) \leftarrow p(f(x))$ we have $p(f(f(...))) \in [p(f(f(...)))] \cap gfp(\mathfrak{T})$, but $p(f(f(...)))$ is not computable at infinity. □

Part (i) of theorem (4.12) establish the soundness of infinite fair derivations: since $gfp(\mathfrak{T})$ satisfies $\mathfrak{T}(gfp(\mathfrak{T})) \subseteq gfp(\mathfrak{T})$, it induces a model of P. In this model each root of an infinite fair derivation is valid. Part (ii) of the theorem states that we cannot hope for a completeness result (in this setting). In general, not every infinite atom in $gfp(\mathfrak{T})$ is the root of an infinite and fair derivation.

EPILOG

Four articles form the basis of our paper. First, Stepwise development of operational and denotational semantics for Prolog by Jones and Mycroft, [26]. Besides their ideas we have taken over their notation. Second, we have used the report of Shapiro about a subset of Concurrent Prolog, [44] to describe Concurrent Prolog in chapter 3. Third, from the article of De Bakker c.s.,[9] we have the notion of transition systems. Finally, chapter 4 is in essence "On the interpretation of infinite computations in logic programming" by Nait Abdallah, [39].

In chapter 2 we constructed a framework of derivations. In this framework we are able to treat renamings in more detail than Apt and Van Emden [1], Van Emden and Kowalski [23], or Lloyd [35]. Moreover we give more proofs. New is the denotational semantics of logic programming, with the use of sets of answer substitutions as domain.
In chapter 3 we extended the ideas of chapter 2 to multisets with the use of a transition system. We give full proof of the equivalence of the operational and denotational semantics of a subset of Prolog. Also new is the operational model of Concurrent Prolog.
In chapter 4 we worked out Nait Abdallah, [39] chapter 4 of Lloyd [35] in detail. We have eliminated all the topology.

Some directions for further research:
Adding to logic programming the possibility to change the set of clauses during execution time, (e.g. Prolog's assert and retract).
Denotational semantics for Concurrent Prolog.
Denotational semantics for logic programming with infinite computations.

REFERENCES

[1] K.R. APT, M.H. VAN EMDEN, *Contributions to the theory of logic programming,* Journal of the ACM 29, 1982, pp. 841-862.

[2] A. ARNOLD, M. NIVAT, *The metric space of infinite trees. Algebraic and topolocical properties,* Fundamenta informaticae III. 4, 1980, pp. 445-476.

[3] R.J. BACK, *A continuous semantics for unbounded nondeterminism,* Theoretical Computer Science 23, 1983, pp. 187-210.

[4] J.W. DE BAKKER, *Mathematical theory of program correctness,* Prentice-Hall, 1980.

[5] J.W. DE BAKKER, J.A. BERGSTRA, J.W. KLOP, J.-J.CH. MEYER, *Linear time and branching time semantics for recursion with merge,* in Proc. 10th ICALP, LNCS 154, J. Diaz (ed.), Springer, 1983, pp. 39-51.

[6] J.W. DE BAKKER, J.I. ZUCKER, *Processes and the denotational semantics of concurrency,* Information and Control 54, 1982, pp. 70-120.

[7] J.W. DE BAKKER, J.I. ZUCKER, *Compactness in semantics for merge and fair merge,* in Proc. Workshop Logics of Programs, LNCS 164, E. Clarke, D. Kozen, (eds.), Springer, 1985, pp. 18-33.

[8] J.W. DE BAKKER, J.I. ZUCKER, *Processes and a fair semantics for the ADA rendez-vous,* Proc. 10th. ICALP, LNCS 154, J. Diaz (ed.), Springer, 1983, pp. 25-39.

[9] J.W. DE BAKKER, J.-J.CH. MEYER, E.-R. OLDEROG, J.I. ZUCKER, *Transition Systems, Infinitary Languages and the Semantics of Uniform Concurrency,* in Proc. 17th ACM STOC, Providence R.I., 1985.

[10] J.W. DE BAKKER, J.N. KOK, *Towards a uniform topological treatment of streams and functions on streams,* in Proc. 12th ICALP, LNCS 194, W. Brauer (ed.), Springer, 1985, pp. 140-148.

[11] J.W. DE BAKKER, J.-J.CH. MEYER, E.-R. OLDEROG, *Infinite Streams and Finite Observations in the Semantics of Uniform Concurrency,* in Proc. 12th ICALP, LNCS 194, W. Brauer (ed.), Springer, 1985, pp. 149-157.

[12] J.W. DE BAKKER, J.-J.CH. MEYER, J.I. ZUCKER, *Bringing color into the semantics of nondeterministic dataflow, preprint.*

[13] J. BERGSTRA, J.W. KLOP, *Process Algebra for Synchronous Communication,* Information and Control 60, 1984, pp. 109-137.

[14] K.L. CLARK, S. GREGORY, *A relational languge for parallel programming,* Proc. 1981 Conference on Functional Programming Languages and Computer Architecture, Porthmouth 1981, pp. 172-178.

[15] P. COUSOT, R. COUSOT, *Constructive versions of Tarski's fixed point theorems,* Pacific Journal of Mathematics 82, 1979, pp. 43-57.

[16] W.F. CLOCKSIN, C.S. MELLISH, *Programming in Prolog,* Springer, 1981.

[17] A. COLMERAUER, *Prolog and infinite trees,* in Logic Programming, K.L. Clark and S.-A. Tärnlund (eds.), Academic Press 1982.

[18] J.S. CONERY, D.F. KIBLER, *Parallel interpretation of logic programs,* Proc. 1981 Conference on Functional Programming Languages and Computer Architecture, Porthmouth 1981, pp. 163-170.

[19] P. DEGANO, U. MONTANARI, *Liveness properties as Convergence in Metric Spaces,* Proc. 16th ACM SICACT Symposium on Theory of Computing, Washington D.C. 1984, pp. 31-38.

[20] J. DUGUNDJI, *Topology,* Allyn and Bacon Inc, 1966.

[21] R. ENGELKING *General Topology,* Polish Scientific Publishers, 1977.

[22] M.H. VAN EMDEN, G.J. DE LUCENA FILHO, *Predicate logic as a language for parallel programming,* in Logic Programming, K.L. Clark and S.-A. Tärnlund (eds.), Academic Press 1982.

[23] M.H. VAN EMDEN, R.A. KOWALSKI, *The semantics of predicate logic as a programming language,* Journal of the ACM 23, pp. 733-742, 1976.

[24] F. HAUSDORFF, *Set theory,* Chelsea 1962.

[25] C.A.R. HOARE, *Communicating Sequential Processes,* Journal of the ACM 21, 1978, pp 666-677.

[26] N.D. JONES, A. MYCROFT, *Stepwise development of operational and denotational semantics for Prolog,* 1984 International symposium on logic programming, Atlantic City, 1984, pp. 281-288.

[27] R.A. KOWALSKI, *Predicate logic as a programming language,* in Proc. IFIP congress, Stockholm, 1974, pp. 569-574.

[28] K. KURATOWSKI, *Topologie, 3me éd,* Polish Scientific Publishers, 1961.

[29] R. KUIPER, *An operational semantics for bounded nondeterminism equivalent to a denotational one,* in Algorithmic Languages, J.W. de Bakker, J.C. van Vliet (eds.), North Holland 1981, pp. 373-398.

[30] J.-L. LASSEZ, M.J. MAHER, *Closures and fairness in the semantics of programming logic,* Theoretical Computer Science 29, 1984, pp. 167-184.

[31] J.-L. LASSEZ, M.J. MAHER, *Optimal fixedpoints of logic programs,* preprint.

[32] G. LEVI, *Processes, Concurrency and Synchronization in Logic Programming,* preprint.

[33] D.W. LOVELAND, *Automated Theorem Proving: a logical basis,* North Holland, 1978.

[34] G. LINDSTROM, P. PANAGADEN, *Stream based execution of logic programs,* in Proc. of the 1984 International Symposium on Logic Programming, Atlantic City 1984, pp. 168-177.

[35] J.W. LLOYD, *Foundations of logic programming,* Springer, 1984.

[36] E. MENDELSON, *Introduction to mathematical logic,* 2nd ed, Van Nostrand, 1979.

[37] J.-J.CH. MEYER, *Fixed points and the arbitrary and fair merge of a fairly simple class of processes,* rapport nr IR-89/IR-92, Free University Amsterdam, 1984.

[38] A. MYCROFT, *Logic programs and many-valued logic,* in Proc. STACS, LNCS 166, M. Fontet, K. Melhorn (eds.), Springer 1984, pp. .

[39] M.A. NAIT ABDALLAH, *On the interpretation of infinite computations in logic programming,* Proc. 11th ICALP, LNCS 172, J. Paredaens (ed.), Springer, 1984, pp. 358-370.

[40] H. NAKAGAWA, *AND Parallel Prolog with Divided Assertion Set,* in Proc. of the 1984 International Symposium on Logic Programming, Atlantic City 1984, pp. 22-28.

[41] M. NIVAT, *Infinite words, infinite trees, infinite computations.* in Foundations of Computer Science III, 2, Mathematical Centre Tracts 109, J.W. de Bakker, J. van Leeuwen (eds.), Mathematical Centre, 1981.

[42] J.A. ROBINSON, *A machine-oriented logic based on the resolution principle,* Journal of the ACM 12, 1965, pp. 23-41.

[43] P. ROUSSEL, *Prolog: Manuel de Reference te d'Utilization,* Université d'Aix-Marseille, 1975.

[44] E. SHAPIRO, A. TAKEUCHI, *Object oriented programming in Concurrent Prolog,* New Generation Computing 1, 1983, pp. 25-48.

[45] E. SHAPIRO, *A subset of Concurrent Prolog and its interpreter,* Department of Applied Mathematics, The Weizmann Institute of Science, 1983.

[46] N. TAMURA, Y. KANEDA, *Implementing parallel Prolog on a multi-processor machine,* in Proc. of the 1984 International Symposium on Logic Programming, Atlantic City 1984, pp. 42-48.