Semantics for Logic Programs without Occur Check

W.P. Weijland
Centre for Mathematics and Computer Science,
Kruislaan 413, 1098 SJ Amsterdam.

abstract: For reasons of efficiency, in almost all implementations of Prolog the *occur check* is left out. This mechanism should protect the program against introducing *circular bindings of variables*. In practice the occur check is very expensive, however, and it is left to the skills of the user, to avoid these circular bindings in the program. In this paper a semantics of Prolog without occur check is introduced. The new kind of resolution, i.e. SLD-resolution without occur check, is referred to as *CSLD*-resolution. Important theorems such as soundness and completeness of both CSLD-resolution and the 'negation as failure' rule, are established.

1. INTRODUCTION

For reasons of efficiency, in almost all implementations of Prolog the *occur check* is left out, which is a mechanism that should protect the program against introducing *circular bindings of variables*. For instance in a substitution {x/f(x)}, the variable x is bound to a term f(x) containing the variable x again. The problem is, that any such binding endangers the correct behaviour of a Prolog system. In fact, without the occur check we no longer have soundness of SLD-resolution (see LLOYD [12]). For example consider the program P: test ← p(x,x)

p(y,f(y)) ← .

Given the goal ← test, a Prolog system without occur check will answer 'yes' since p(x,x) will be successfully unified with p(y,f(y)) by the substitution {x/y, y/f(y)}. However, this answer is quite wrong, since test is not a logical consequence of P.

In practice, however, the occur check is very expensive and it is usually left to the skills of the user to avoid these circular bindings in the program. For instance in PLAISTED [14], a method is presented to detect circular bindings more efficiently, by preprocessing Prolog programs.

It would be convenient to develop a theory for SLD-resolution without occur check, and for this reason Prolog II (see COLMERAUER [3]) has been studied quite intensively in the past few years. Roughly speaking, Prolog II is standard Prolog without occur check and can be regarded as a system which manipulates infinite trees (see COLMERAUER [2]).

The question remains, whether or not Prolog II can be thought of as a logic programming language, since the example above shows that Prolog II presents incorrect derivations. This problem was solved by VAN EMDEN & LLOYD [6], by formulating a soundness theorem for Prolog II. In the above example, the computed substitution {x/y,y/f(y)} can be translated to a set of equations {x=y,y=f(y)}, and clearly test is a logical consequence of P∪{x=y,y=f(y)}. There are still many results left to be established, such as completeness for instance, to develop a complete theory for Prolog II.

In this paper a semantics for logic programs without occurcheck is presented by considering circular bindings {x/f(x)} as *recursive equations* {x=f(x)}, and extending the Herbrand universe (consisting of all closed terms) by adding all infinite terms {x=f(f(f(...)))} to it (see COURCELLE [4]). We introduce a new kind of resolution, which will be referred to as CSLD-resolution (*complete* SLD-resolution), which is precisely SLD-resolution without occur check. Following this idea, we find that both soundness and completeness for CSLD-resolution as well as for the negation as failure rule is obtained. It turns out that due to the new setting, the proof of the completeness theorem for the negation as failure rule becomes much shorter compared with the well-known proofs in [9], [12] and [15]. Then, we conclude as a general result that comp(P)$\cup${A} has a 'complete' Herbrand model iff it has a model, which indicates that we may expect CSLD-resolution to have some nice properties extra, that we do not have for ordinary SLD-resolution.

Independently from this paper, similar results were stated by JAFFAR & LASSEZ [7] and MAHER [13], on the 'Constraint Logic Programming' scheme, JAFFAR, LASSER & MAHER [8], on PROLOG II as a Logic Programming Language scheme, and JAFFAR & STUCKEY [10], giving a logical semantics to a language without occur check containing both equations and inequations.

There are a few theoretical differences between these references and the contents of this paper. Consider for example the fact that we will only need a small equational theory for Prolog II, whereas in [6] and [8] this theory contains infinitely many existential formulas, one for every recursive equation. For this reason we do not need to put any constraints on the models of Prolog II programs and the results are more general. Still, apart from the question whether or not the main results are new, we believe to have found a rather elegant formalisation of the theory, making the techniques used in this paper of interest by themselves. The concepts and notations in this paper are quite similar to those from LLOYD [12], which may be considered a contribution to standardising the theory of logic programming without occur check.

2. COMPLETE HERBRAND MODELS

We will assume P to be a set of *program clauses* $\forall(B_1\wedge\ldots\wedge B_k\rightarrow A)$, usually written as $A\leftarrow B_1,\ldots,B_k$, where $B_1,\ldots,B_k,A$ are atoms not containing '='. The language of P will be denoted by L(P) or L_P.

In this section we will formally introduce *complete* Herbrand models for P. First we will present a precise definition of a *complete term*, as can be found in [12], and next establish some general model theoretical results.

Let ω^* be the set of all finite sequences of non-negative integers. Such a finite sequence will be written as $[i_1,\ldots,i_k]$, for some $i_1,\ldots,i_k\in\omega$. For all $m,n\in\omega^*$ we write $[m,n]$ for the concatenation of m and n, and for $i\in\omega$ we write $[m,i]$ instead of $[m,[i]]$. For $X\in\omega^*$ we write $|X|$ for the cardinality of X.

definition 2.1 $T\subseteq\omega^*$ is called a *tree* if T satisfies the following conditions:

(i) for all $n\in\omega^*$ and $i,j\in\omega$: $[n,i]\in T\wedge j<i\Rightarrow n\in T\wedge[n,j]\in T$

(ii) $|\{i:[n,i]\in T\}|$ is finite for all $n\in T$.

So, by definition 2.1 we can interpret [ ] as the *root* of the tree and $[n,0],[n,1],\ldots,[n,k]$ as the descendents of the node n for all $n\in T$, $k<\omega$.

Now let S be a set of symbols and ar: $S\rightarrow\omega$ be a mapping defining the *arity* of a symbol.

definition 2.2 A *complete term* (over S) is a function t: $\mathrm{dom}(t)\rightarrow S$ such that:

(i) the domain of t, dom(t), is a non-empty tree

(ii) for all $n\in\mathrm{dom}(t)$: $ar(t(n))=|\{i:[n,i]\in\mathrm{dom}(t)\}|$.

In a language L, a *complete atom* is a complete term t such that t([]) is a predicate symbol.

definition 2.3 The *depth* dp(t) of a term t is defined by:

(i) $dp(t) = \infty$, if t is infinite

(ii) $dp(t) = 1 + \max\{|n| : n \in dom(t)\}$, if t is finite.

The tree dom(t) is called the *underlying tree* of t. The set of all complete terms over S is denoted by $Terms_S$; these terms can be looked at as (possibly) infinite terms. By definition a term t is finite if and only if dom(t) is finite. Next, we will define a metric on $Terms_S$.

definition 2.4 Let $s,t \in Terms_S$ and $s \neq t$ then we define $\beta(s,t)$ as being the *least* depth at which s and t differ. Then we define: (i) $d(s,t) = 0$ if $s=t$

(ii) $d(s,t) = 2^{-\beta(s,t)}$ if $s \neq t$.

proposition 2.1 ($Terms_S$, d) is a (ultra-)metric space.

The proof is simple, and omitted here. Note that the larger the depth is at which two terms differ, the smaller is their distance. Next, we define the *truncation* of a term, to have finite approximations of infinite terms. Assume Ω to be an extra constant symbol (hence with arity zero), not in S (nor in $Terms_S$).

definition 2.5 The *truncation at depth n* of a term t, notation $\alpha_n(t)$, can be found from the complete term t by replacing all symbols at depth n by Ω and leaving out all symbols at greater depth. The underlying tree $dom(\alpha_n(t))$ is adjusted in the same way, by leaving out all nodes without a label.

definition 2.6 A metric space (X,d) is *compact* if every sequence in X has a subsequence which converges to a point in X.

proposition 2.2 ($Terms_S$,d) is compact iff S is finite.

For a proof of this wellknown theorem, see [12]. From definition 2.5 we find: $dp(\alpha_n(t)) \leq n+1$, for all t. Moreover, $d(\alpha_n(t),t) \leq 2^{-n}$ and therefore $\lim_{n\to\infty} \alpha_n(t) = t$. Next we will consider complete Herbrand models for a program P, having all possibly infinite terms as its universe.

definition 2.7 Let P be a program, then the *complete Herbrand universe* CU_P is defined by $Term_{L(P)}$. A *complete Herbrand model* for L_P is a model $\mathcal{M}$ with domain CU_P, such that

(i) $a^{\mathcal{M}} = a$, for all constants $a \in L_P$

(ii) $f^{\mathcal{M}}(t_1,...,t_k) = f(t_1{}^{\mathcal{M}},...,t_k{}^{\mathcal{M}})$, for all functions $f \in L_P$ and complete terms $t_1,...,t_k \in Term_{L(P)}$.

A complete Herbrand model for a program P is a complete Herbrand model for L_P which satisfies P.

definition 2.8 The *complete Herbrand base* CB_P, or $CB_{L(P)}$, of a program P is defined by

$CB_P = \{R(t_1,...,t_k) : R \text{ is a relation symbol in } L_P \text{ and } t_1,...,t_k \in CU_P\}$.

The elements of CB_P can be represented as trees as well. Moreover, the metric d on CU_P can be extended to CB_P. Informally, we will write U_P for the set of all finite terms in L_P and B_P for the set of all finite elements from CB_P.

In general any complete Herbrand model for a program P, can be associated with a subset of the complete Herbrand base CB_P: such a subset then denotes the complete set of 'ground' atoms, holding in the model. For any ground atom A and Herbrand model $\mathcal{M}$, we will use both notations $A \in \mathcal{M}$ and $\mathcal{M} \models A$, to express that A holds in the model $\mathcal{M}$ and $\models A$ to express that A holds in all (possibly non-Herbrand) models.

3. RECURSIVE SPECIFICATIONS

In this section we consider so called recursive specifications, which are finite sets of positive equational formulas and will be used later instead of the usual notion of a *substitution*. Returning to Prolog, we will need a different *unification algorithm*, since we will work in *complete* Herbrand models.

definition 3.1 Let P be a program. Then the theory $Eq(L_P)$ (or Eq(P)) consists of the axioms:

1. $c \neq d$	for all pairs of distinct constants c,d from L_P
2. $\forall x.\ f(x) \neq g(x)$	for all pairs of distinct function symbols f,g from L_P
3. $\forall x.\ f(x) \neq c$	for all function symbols f and all constants c from L_P
4. $\forall xy.\ x_1 \neq y_1 \vee \ldots \vee x_k \neq y_k \rightarrow f(x) \neq f(y)$	for all function symbols f from L_P
5. $\forall x.\ x = x$	
6. $\forall xy.\ x_1 = y_1 \wedge \ldots \wedge x_k = y_k \rightarrow f(x) = f(y)$	for all function symbols f from L_P
7. $\forall xy.\ x_1 = y_1 \wedge \ldots \wedge x_k = y_k \rightarrow (P(x) \rightarrow P(y))$	for all predicate symbols P from L_P.

lemma 3.1 Eq(L) holds in all complete Herbrand models for L.

The axioms of $Eq(L_P)$ are introduced in [12] to model finite failure: $Eq(L_P)$ forces any two syntactically different terms to be different in all its models. In [12] we even find an extra axiom:

8. $\forall x.\ x \neq t[x]$

for all terms t that are unequal to a variable and contain the variable x. This axiom is needed to express that the elements in the Herbrand universe consist of all *finite* terms from L_P. Since in the complete Herbrand universe we do have infinite terms as well, we will omit axiom 8 from our equational theory.

It turns out to be convenient to consider substitutions no longer as a syntactical operation of binding variables, but directly as equational formulas.

definition 3.2 A *(recursive) specification* in a language L is a set of equations of the form: $\{t_1(x)=s_1(x),\ldots,t_k(x)=s_k(x)\}$ for (open) terms $t_i, s_i \in L$ and variables $x \equiv x_1,\ldots,x_n$.

definition 3.3 An *open* complete term in a language L is obtained by constructing a complete term from $L \cup \{x_i : i \leq n\}$, where $\{x_i : i \leq n\}$ denotes a finite set of variable symbols with arity zero.

Note that an open complete term only has finitely many variables, called the *free* variables of the term. We will write $t(\mathbf{x})$ for the term t which has variables only from $\mathbf{x} \equiv x_1,...,x_n$ (but possibly less) and similarly we write $\rho(\mathbf{x})$ for a specification with variables only from $\mathbf{x} \equiv x_1,...,x_n$.
Note that the metric d can simply be generalised to open complete terms, by extending the language L with extra variable symbols.

proposition 3.2 Let L be a language with at least one constant, and let $\mathcal{M}$ be a complete Herbrand model for L. Then for all open complete terms $t_1(\mathbf{x})$, $t_2(\mathbf{x})$ we have:
$(\forall \mathbf{s} \in U_L: \mathcal{M} \models t_1(\mathbf{s}) = t_2(\mathbf{s})) \Leftrightarrow d(t_1(\mathbf{x}),t_2(\mathbf{x})) = 0.$

definition 3.4 A specification $\rho(\mathbf{x})$ is said to be in *reduced form* if it is of the form $\{x_1=s_1(x_1,...,x_n),...,x_k=s_k(x_1,...,x_n)\}$, where $x_1,...,x_k$ are distinct variables. Moreover, $\rho(\mathbf{x})$ *has* a reduced form if it is equivalent to a specification which is in reduced form.
A specification is said to be in *contradictory form* if it contains an equation a=b or $f(t_1,...,t_n)=g(s_1,...,s_n)$ for some distinct symbols a,b or f,g respectively. Moreover, it *has* a contradictory form if it is logically equivalent to a specification which is in contradictory form.

definition 3.5 A variable x is *bound* in ρ if $\rho \models x=t$ for some term t which is not a variable. Otherwise it is called *free*. A specification is called *ground* if it has no free variables.

example Let $\sigma(x,y) = \{x=f(x), y=x\}$, then σ has no free variables since $\sigma \models x=f(x)$ and $\sigma \models y=f(x)\}$.
Let $\sigma(x,y,z) = \{x=f(y), y=z\}$, then x is a bound variable in σ, whereas y and z are free.

definition 3.6 A specification $\rho(\mathbf{x})$ is called *consistent* if $\rho \cup Eq(\rho)$ is satisfiable in a model.

theorem 3.3 Let L be a language. If $\rho(\mathbf{x},\mathbf{y})$ is in reduced form, with bound variables $\mathbf{x} \equiv x_1,...,x_n$ which are distinct and free variables $\mathbf{y}$, then there exist (open) complete terms $t_1(\mathbf{y}),...,t_n(\mathbf{y})$ with only *free* variables from ρ, such that in every complete Herbrand model $\mathcal{M}$ for L:
$$\mathcal{M} \models \forall \mathbf{xy}.\ \rho(\mathbf{x},\mathbf{y}) \leftrightarrow (x_1=t_1(\mathbf{y}) \wedge ... \wedge x_n=t_n(\mathbf{y})).$$

proof Use the fact that for all equations $x=t(x,\mathbf{y})$ in ρ: $\mathcal{M} \models \forall x\mathbf{y}.\ x=t(x,\mathbf{y}) \leftrightarrow x=z\ [z/t^{\omega}(\mathbf{y})]$, in every complete Herbrand model $\mathcal{M}$ for L_ρ and with $t^{\omega}(\mathbf{y}):=t(t(t(...,\mathbf{y}),\mathbf{y}),\mathbf{y})$. □

theorem 3.4 (i) A specification in reduced form is consistent.
(ii) A specification in contradictory form is inconsistent.

proof (i) Suppose ρ is in reduced form, with language L. Extend L until it contains at least one constant, then L has a complete Herbrand model $\mathcal{M}$. Then from theorem 3.3 and lemma 3.1 it follows that: $\mathcal{M} \models \rho \cup Eq(\rho)$, thus ρ is consistent. Part (ii) follows directly from the definition of $Eq(\rho)$. □

corollary 3.5 Every specification in reduced form with a constant, has a complete Herbrand model.

theorem 3.6 (i) All consistent specifications have a reduced form.
(ii) All inconsistent specifications have a contradictory form.

The proof of corollary 3.5 follows from the proof of theorem 3.4. Theorem 3.6 is the reverse of theorem 3.4. In order to prove it, an algorithm can be constructed which actually *decides* whether a specification is consistent or not, by calculating an equivalent specification in reduced form (if such specification exists). This algorithm consists of the following five steps, defined in COLMERAUER [2] (see also [8]). Suppose ρ is a specification.

consistency algorithm

(1) Delete from ρ all equations of the form $x=x$.
(2) If ρ contains an equation $x=y$, where x and y are different variables, then replace x in all its occurences in ρ by y.
(3) Replace an equation $t=x$ in ρ by $x=t$, where t is not a variable.
(4) Replace two equations $x=t$ and $x=s$ in ρ by the equations $x=t$ and $t=s$, where t is the smaller (in number of symbols) of the two terms t and s.
(5) Replace an equation of the form $f(t_1,\ldots,t_n)=f(s_1,\ldots,s_n)$ by the equations $t_1=s_1,\ldots,t_n=s_n$.

It is well-known that, repeatedly using these five steps, any recursive specification ρ can be reduced into either a reduced form or a contradictory form, which is equivalent to ρ. This provides us with a proof of theorem 3.6.
Furthermore, using the consistency algorithm one can define a new kind of unification which precisely coincides with unification in Prolog II. Assume p is a predicate symbol and S is a set of atoms.

unification algorithm

(1) If not all atoms in S start with the same predicate symbol, then S is not unifiable.
(2) Else: if $S = \{p(t^{(i)}_1,\ldots,t^{(i)}_n): i \leq m\}$ then apply the consistency algorithm to $\{t^{(1)}_1=t^{(2)}_1, t^{(2)}_1=t^{(3)}_1,\ldots,t^{(m-1)}_1=t^{(m)}_1,\ldots,t^{(1)}_n=t^{(2)}_n, t^{(2)}_n=t^{(3)}_n,\ldots,t^{(m-1)}_n=t^{(m)}_n\}$.

corollary 3.7 A specification ρ is consistent iff it has a reduced form.

definition 3.7 Let S be a set of (open) atoms.
A specification ρ is called *complete unifier* (cu) for S, if: $\models \forall(\rho \rightarrow \bigwedge_{A,B \in S} (A \leftrightarrow B))$.
Suppose ρ is a cu for S. ρ is called *most general complete unifier* (mcu) for S, if for all complete unifiers ρ_1 for S and all complete Herbrand models $\mathcal{M}$: $\mathcal{M} \models \forall(\rho_1 \rightarrow \rho)$.

proposition 3.8 For any input set S of atoms, the unification algorithm computes an mcu for S.

Note that proposition 3.8 does not hold if we change definition 3.7 by requiring $\forall(\rho_1 \rightarrow \rho)$ to hold in *all* models instead of only complete Herbrand models. For instance if $S = \{p(x), p(f(f(x))\}$ then $\{x=f(f(x))\}$ is computed by the unification algorithm, although $\{x=f(x)\}$ is more general. With respect to complete

Theorem 4.8 is part of a more general result from JAFFAR & STUCKEY [10] on a language with equations and inequations.
So far, we found that the correctness theorem can be restored for CSLD-resolution. Moreover, $CT_P\uparrow\omega$ is the least complete Herbrand model which is the intersection of all complete Herbrand models for P, and equal to the complete success set. Next we will show we have a completeness theorem as well.

definition 4.10 (restriction) Let $\sigma(\mathbf{x},\mathbf{y})$ and $\rho(\mathbf{x},\mathbf{z})$ be two specifications then we write $\sigma\lesssim_{\mathbf{x}}\rho$ if $\models\forall \mathbf{xy}.(\sigma(\mathbf{x},\mathbf{y})\rightarrow\exists \mathbf{z}.\rho(\mathbf{x},\mathbf{z}))$. Furthermore we write $\sigma\equiv_{\mathbf{x}}\rho$ if both $\sigma\lesssim_{\mathbf{x}}\rho$ and $\rho\lesssim_{\mathbf{x}}\sigma$.

proposition 4.9 Let $\sigma(\mathbf{x})$ be ground then for all specifications ρ: either $\sigma\rho\equiv\perp$ or $\sigma\lesssim_{\mathbf{x}}\rho$.

Definition 4.10 is needed to indicate that σ is more specific than ρ, although ρ may bind variables not occurring in σ. Moreover, $\sigma\equiv_{\mathbf{x}}\rho$ indicates that σ and ρ are equivalent with respect to the variables $\mathbf{x}$. Note that $\sigma\equiv_{\mathbf{x}}\perp \Rightarrow \sigma\equiv\perp$ for all variables $\mathbf{x}$. Proposition 4.9 says that *ground* specifications cannot be further specified with respect to their variables: either $\sigma\cup\rho$ is inconsistent, or σ is more specific than ρ with respect to $\mathbf{x}$. For example: let $\sigma(x)=\{x=f(x,x)\}$ and $\rho(x,y)=\{x=f(x,y),\ y=x\}$ then $\sigma\lesssim_{x}\rho$, however not $\sigma\lesssim\rho$ (i.e.: $\sigma\lesssim_{xy}\rho$), since ρ has an extra variable y. In fact: $\sigma\equiv_{x}\rho$.

lemma 4.10 Let $A(\mathbf{x})$ be an atom and $\sigma(\mathbf{x})$ correct for $P\cup\{\leftarrow A(\mathbf{x}),\emptyset\}$, then there exists a CSLD-refutation for $P\cup\{\leftarrow A(\mathbf{x}),\sigma(\mathbf{x})\}$ with $\rho(\mathbf{x},\mathbf{y})$ as computed answer specification, such that $\sigma\equiv_{\mathbf{x}}\rho$.

proof First, assume $\sigma(\mathbf{x})$ is ground. Now, let $\mathbf{u}\in\sigma$ then $A(\mathbf{u})\in CT_P\uparrow\omega$ (by theorem 3.9, definition 4.6 and corollary 4.5). Therefore $A(\mathbf{u})\in CS_P$ (by theorem 4.8), hence there is a CSLD-refutation $(G_i,\rho_i)_{0\leq i\leq n}$ for $P\cup\{\leftarrow A(\mathbf{x}),\emptyset\}$ with computed answer specification $\rho_n(\mathbf{x},\mathbf{y})$ such that for some $\mathbf{v}$: $(\mathbf{u},\mathbf{v})\in\rho_n$. Because $(\mathbf{u},\mathbf{v})\in\sigma\rho_n$, $\sigma\rho_n$ is consistent and therefore $(G_i,\sigma\rho_i)_{0\leq i\leq n}$ is a refutation for $P\cup\{\leftarrow A(\mathbf{x}),\sigma\}$ with computed answer specification $\sigma\rho_n$, and by proposition 4.1 we have $\sigma\rho_n\lesssim\sigma$. Since σ is ground we find by proposition 4.9: $\sigma\lesssim_{\mathbf{x}}\sigma\rho_n$. Hence $\sigma\equiv_{\mathbf{x}}\sigma\rho_n$.
Next, assume σ is not ground, and let $\mathbf{x}=(\mathbf{y},\mathbf{z})$ where $\mathbf{y}$ and $\mathbf{z}\equiv z_1,\ldots,z_k$ are the bound and free variables respectively of σ. Let $\mathbf{a}\equiv a_1,\ldots,a_k$ be *new* constants not occurring in P,A or σ, and such that for i,j: $a_i=a_j \Leftrightarrow \sigma\models z_i=z_j$. Next, consider $\sigma'(\mathbf{y},\mathbf{z})\equiv\{z_1=a_1,\ldots,z_n=a_n\}\cdot\sigma(\mathbf{y},\mathbf{z})$, then it is easily proved that σ' is consistent and ground. Hence there exists a CSLD-refutation for $P\cup\{\leftarrow A,\sigma'(\mathbf{y},\mathbf{z})\}$ with computed answer specification $\rho'(\mathbf{y},\mathbf{z},\mathbf{z}')$ such that $\sigma'\equiv_{\mathbf{yz}}\rho'$, or equivalently: $\sigma'\equiv_{\mathbf{x}}\rho'$. Now it is easy to see, that we can find a new refutation for $P\cup\{\leftarrow A,\sigma\}$ by replacing all constants $\mathbf{a}$ by new variables, with computed answer specification ρ, such that $\sigma\equiv_{\mathbf{x}}\rho$. □

Note that lemma 4.10 does not hold if we replace $\equiv_{\mathbf{x}}$ by $\equiv$. For example let $P = \{A(y,y)\leftarrow\}$ and consider the goal clause $G=\ \leftarrow A(x,f(x))$ then one can easily see that $\sigma(x)=\{x=f(x)\}$ is correct for $P\cup\{G,\emptyset\}$. Indeed, there is a computed answer specification $\rho(x,y)=\{x=\{y,f(y)\}\}$ which is equivalent to $\rho'(x,y)=\{x=y,\ y=f(y)\}$. Clearly $\rho\lesssim\sigma$, however, since $\not\models\ \forall(\ x=f(x)\rightarrow(x=y\wedge y=f(y))\ )$ the converse is not true. Hence $\rho\not\equiv\sigma$.

The point is, that in a CSLD-refutation new variables are introduced (input clauses have new variables), and the correct specification we started with cannot impose any constraints upon variables other than its own. In 'common' SLD-resolution, this problem does not occur since computed substitutions are restricted to the goal variables automatically. This can be done, because in SLD-resolution new variables are bound to finite terms (not containing the variable again), hence one can simply carry out the substitution. This problem is overcome, however, by introducing $\lesssim_x$ as a logical notion of restriction.

lemma 4.11 (lifting lemma) Let G be a goal clause and σ be a specification. Assume there exists a CSLD-refutation for $P\cup\{G,\sigma\}$ with computed answer specification ρ. If $\sigma\lesssim\sigma'$ then there is a CSLD-refutation for $P\cup\{G,\sigma'\}$ with computed answer specification ρ' of the same length such that $\rho\lesssim\rho'$.

theorem 4.12 (completeness of CSLD-resolution) Let $(G_0(x),\sigma_0(x))$ be a goal and $\sigma(x)$ a correct answer specification for $P\cup\{G_0(x),\sigma_0(x)\}$ then there exists a computation rule R and a R-computed answer specification $\rho(x,y)$ for $P\cup\{G_0(x),\sigma_0(x)\}$ such that $\sigma\lesssim_x\rho$.

proof Assume $G_0=(\leftarrow A_1(x),\dots,A_k(x))$ then $P\models\forall(\sigma\rightarrow A_1(x)\wedge\dots\wedge A_k(x)\wedge\sigma_0)$ since σ is correct for $P\cup\{G_0,\sigma_0\}$. By lemma 4.10 there exist refutations for $P\cup\{\leftarrow A_i,\sigma\}$ with ρ_i as computed answer specification, such that $\rho_i\equiv_x\sigma$ for all i. These refutations can be combined to obtain a new refutation for $P\cup\{\leftarrow A_1(x),\dots,A_k(x),\sigma\}$ with answer specification $\rho'\equiv\rho_1\cdots\rho_k$, so $\sigma\equiv_x\rho'$. Since $\sigma\lesssim\sigma_0$, it follows by the lifting lemma that there exists a CSLD-refutation for $P\cup\{\leftarrow A_1(x),\dots,A_k(x),\sigma_0\}$ with computed answer specification ρ such that $\rho'\lesssim\rho$. Since $\sigma\equiv_x\rho'$ we have $\sigma\lesssim_x\rho$. □

The proof of the lemma 4.11 is similar to the one in [12]. The completeness theorem presented above, can be obtained from MAHER [13] and some additional remarks when applied to the axioms of JAFFAR, LASSEZ & MAHER [8].

It is important to understand how resolution with specifications works. In fact, a computed answer specification can be looked at as an extra condition or *constraint* (see also [7]) that needs to be satisfied before a given conclusion may be drawn from P. The completeness theorem simply states that from a logic program all such sufficient conditions can be generated.

The completeness theorem leads almost directly to the following corollary:

corollary 4.13 Let A(x) be a atom and $\mathbf{u}\equiv u_1,\dots,u_k$ complete terms, then:

$A(u)\in CS_P \Leftrightarrow$ for some $\rho(x,y)$ and some $\mathbf{v}$: $(\mathbf{u},\mathbf{v})\in\rho$ and $P\models\forall(\rho(x,y)\rightarrow A(x))$.

5. FINITE FAILURE

In this section we will consider the *negation as failure rule*, for CSLD-resolution. It turns out that all 'classical' results can be restored; even better: it seems that working in CSLD-semantics can simplify some theoretical constructions. Let us start with some definitions.

definition 5.1 Let G be a goal. A *CSLD-tree* for $P\cup\{G\}$ with c-rule R, is defined by

(i) every node of the tree is a goal and the root is equal to G

(ii) if $G'=(\leftarrow A_1,\dots,A_m,\dots,A_k,\sigma)$ is a node for some consistent σ, and $R(G')=A_m$ then G' has a *successor* $(\leftarrow A_1,\dots,A_{m-1},B_1,\dots,B_q,A_{m+1},\dots,A_k,\sigma\rho)$ for every clause $A\leftarrow B_1,\dots,B_q\in P$, where $\rho\equiv mcu(\{A_m,A\})$; if σ is inconsistent, then G' has no successor goals.

definition 5.2 A *success branch* in a CSLD-tree is a branch that ends with $(\square,\sigma)$ for some consistent specification σ. A *failure branch* is a branch $(G_i,\sigma_i)_{0\le i\le k}$ such that $\sigma_k\equiv\perp$.

definition 5.3 A *finitely failed CSLD-tree*, or *ff-tree* for short, for $P\cup\{G\}$ is a finite CSLD-tree with only failure branches.

definition 5.4 We will use the following notation:

$$CT_P\downarrow 0 = CB_P$$
$$CT_P\downarrow k+1 = CT_P(CT_P\downarrow k)),\ \ k\in\omega$$
$$CT_P\downarrow\omega = \bigcap_{k<\omega} CT_P\downarrow k$$

A well-known theorem says that $CT_P\downarrow\omega$ is the *greatest fixed point*, denoted by $gfp(CT_P)$, of the continuous mapping CT_P. Recall that $T_P\downarrow\omega$ does not need to be equal to the greatest fixed point of T_P.

lemma 5.1 Let A(x) be an atom and $\mathbf{u}\equiv u_1,\dots,u_k$ be complete terms, then

$A(\mathbf{u})\in CT_P\downarrow k+1 \Leftrightarrow$ there is a clause $A_1(\mathbf{y})\leftarrow B_1(\mathbf{y}),\dots,B_q(\mathbf{y})\in P$, such that for some v:
$(\mathbf{u},\mathbf{v})\in mcu(\{A(\mathbf{x}),A_1(\mathbf{y})\})$ and $B_1(\mathbf{v}),\dots,B_q(\mathbf{v})\in CT_P\downarrow k$.

theorem 5.2 Suppose R is a *fair* c-rule, i.e.: every atom in a goal clause is selected somewhere in the CSLD-tree, and assume $(G_i,\sigma_i)_{i\in\omega}$ is an infinite CSLD-derivation for $P\cup\{G_0,\sigma_0\}$ by R with $G_0=(\leftarrow A_1(\mathbf{x}),\dots,A_n(\mathbf{x}),\sigma_0)$. If **u** are complete terms, then:

$$\forall k\ \exists i.\ (\ (\mathbf{u},\mathbf{v})\in\sigma_i(\mathbf{x},\mathbf{y})\ \Rightarrow\ A_1(\mathbf{u}),\dots,A_n(\mathbf{u})\in CT_P\downarrow k\ ).$$

proof Let $k\in\omega$, and **u** complete terms. Let $(G_i,\sigma_i)_{i\in\omega}$ be an infinite derivation for $P\cup(G_0,\sigma_0)$. Then by induction on k: **k=0:** Immediately.

k+1: Let $1\le j\le n$ and let $m\in\omega$ such that $R(G_m)=A_j$. This m exists because R is fair. Then there is some clause $A(\mathbf{y})\leftarrow B_1(\mathbf{y}),\dots,B_q(\mathbf{y})$ such that $\rho(\mathbf{x},\mathbf{y})\equiv mcu(\{A_j(\mathbf{x}),A(\mathbf{y})\})$ is consistent and $\sigma_{m+1}\equiv\sigma_m\cdot\rho$. Assume $G_m=(\leftarrow C_1,\dots,A_j,\dots,C_r)$ then $G_{m+1}=(\leftarrow C_1,\dots,B_1,\dots,B_q,\dots,C_r)$ and clearly $(G_{m+i},\sigma_{m+i})_{i\ge 1}$ is an infinite derivation for $P\cup(G_{m+1},\sigma_{m+1})$. By induction, let $i'\in\omega$ such that $(\mathbf{u},\mathbf{v})\in\sigma_{i'}(\mathbf{x},\mathbf{y})\ \Rightarrow\ C_1(\mathbf{u},\mathbf{v}),\dots,B_1(\mathbf{v}),\dots,B_q(\mathbf{v}),\dots,C_r(\mathbf{u},\mathbf{v})\in CT_P\downarrow k$, then it follows by definition of CT_P that for all $(\mathbf{u},\mathbf{v})\in\sigma_{i'}(\mathbf{x},\mathbf{y})\Rightarrow A(\mathbf{v})\in CT_P\downarrow k+1$. Since $\sigma_{i'}\le\rho$ we have $A_j(\mathbf{u})\in CT_P\downarrow k+1$. So, for every j such an index i' exists. Now take the maximum of all n indices. $\square$

lemma 5.3 If (G,σ) has an ff-tree with depth $\le k$, then $(G,\sigma\rho)$ has an ff-tree with depth $\le k$.

lemma 5.4 If for some *ground* specification $\sigma(x)$, the goal $(\leftarrow A_1(x),\ldots,A_n(x),\sigma(x))$ has an ff-tree with depth $\le k$, then for some $i\le n$: $(\leftarrow A_i(x),\sigma(x))$ has an ff-tree with depth $\le k$.

proof By induction on n: **n=1:** immediately.

n+1: assume $A_{n+1}(x)$ has no ff-tree with depth $\le k$, then for all c-rules R, $P\cup\{A_{n+1}(x),\sigma(x)\}$ has a CSLD-derivation of length $\ge k+1$. Let R be a c-rule such that $P\cup\{\leftarrow A_1(x),\ldots,A_{n+1}(x),\sigma(x)\}$ has an ff-tree with depth $\le k$, and let $(G_i,\rho_i(x,y_i))_{0\le i\le k+1}$ be a derivation for $P\cup\{A_{n+1}(x),\sigma(x)\}$ via R with length $\ge k+1$, then for all derivations $(G'_i,\tau_i(x,z_i))_{0\le i\le m}$ for $P\cup\{\leftarrow A_1(x),\ldots,A_n(x),\sigma(x)\}$ we have (by 4.1(iii) and 4.9) that for all i,j: $\tau_i\rho_j\equiv\perp \Rightarrow \tau_i\equiv\perp \vee \rho_i\equiv\perp$, since $\sigma(x)$ is ground and $\tau_0=\rho_0=\sigma$. Then, there is a finitely failed derivation $(G_i,\theta_i)_{0\le i\le q}$ via R for $P\cup\{\leftarrow A_1(x),\ldots,A_{n+1}(x),\sigma(x)\}$ such that $\theta_i\ge\tau_i\rho_i$ and $q\le k$ (since all derivations via R are finitely failed with length $\le k$) we find that $\tau_q\rho_q\equiv\perp$. Because ρ_q is consistent for all $q\le k$ we have $\tau_q\equiv\perp$.

Therefore, all derivations for $P\cup\{\leftarrow A_1(x),\ldots,A_n(x),\sigma(x)\}$ are finitely failed with length $\le k$, hence $P\cup\{\leftarrow A_1(x),\ldots,A_n(x),\sigma(x)\}$ has an ff-tree with depth $\le k$. Now by induction. □

Lemma 5.3 is easy to prove. Note that in lemma 5.4 σ needs to be ground (see also in [12]).

theorem 5.5 If $A_1(u),\ldots,A_n(u)\in CT_P{\downarrow}k$ for some complete terms $u\in\sigma(x)$, then there exists a *ground* specification $\rho(x,y)\le\sigma(x)$ such that for $(v,w)\in\rho(x,y)$: $A_1(v),\ldots,A_n(v)\in CT_P{\downarrow}k$.

proof By induction on k. Assume $u\in\sigma(x)$, for some specification σ.

k=0: Immediately, since any ground $\rho\le\sigma$ suffices.

k+1: Suppose $A_1(u),\ldots,A_n(u)\in CT_P{\downarrow}k+1$ then by the definition of CT_P there exist clauses $A^i(y_i)\leftarrow B^i_1(y_i),\ldots,B^i_q(y_i)\in P$ for all $i\le n$, such that with $\rho^i(x,y_i)\equiv mcu(\{A_i(x),A^i(y_i)\})$, $\rho\equiv\rho^1\ldots\rho^n$ is consistent. Clearly, $\sigma\rho(x,y_1,\ldots,y_n)$ is consistent as well. Writing $y\equiv y_1,\ldots,y_n$ there exist $v\equiv v_1,\ldots,v_n$ such that $(u,v)\in\sigma\rho(x,y)$ and for all $i\le n$: $B^i_1(v_i),\ldots,B^i_q(v_i)\in CT_P{\downarrow}k$. It follows by induction that there exists a ground specification $\tau(x,y,z)\le\sigma\rho(x,y)$ such that for $(w,w',w'')\in\tau(x,y)$: $B^i_1(w'),\ldots,B^i_q(w')\in CT_P{\downarrow}k$. Since $\rho\le\rho^i$ we have $A_i(w)\equiv A^i(w')$ and therefore $A_1(w),\ldots,A_k(w)\in CT_P{\downarrow}k+1$, by definition of CT_P. Since $\sigma\rho\le\sigma$ we find $\tau\le\sigma\rho\le\sigma$. □

theorem 5.6 If $\sigma(x)$ is ground and $(\leftarrow A_1(x),\ldots,A_n(x),\sigma(x))$ has an ff-tree with depth $\le k$, then for some $i\le n$: $\forall u.\,(\,u\in\sigma(x) \Rightarrow A_i(u)\notin CT_P{\downarrow}k\,)$.

proof By induction on k. **k=1:** directly by lemma 5.1.

k+1: Suppose $A_i(u)\in CT_P{\downarrow}k+1$, for $u\in\sigma$, then there is a clause $A(y)\leftarrow B_1(y),\ldots,B_q(y)\in P$ such that $\rho(x,y)\equiv mcu(\{A_i(x),A(y)\})$ is consistent and for some v: $(u,v)\in\rho$ and $B_1(v),\ldots,B_q(v)\in CT_P{\downarrow}k$ (see lemma 5.1). Clearly $\sigma\rho$ is consistent and $(u,v)\in\sigma\rho(x,y)$. Then, by theorem 5.5 it follows that there is some ground specification $\tau(x,y,z)\le\sigma\rho(x,y)$ such that: $B_1(v'),\ldots,B_q(v')\in CT_P{\downarrow}k$ for all $(u',v',w')\in\tau$. Since τ is ground, it follows by induction that $(\leftarrow B_1(y),\ldots,B_q(y),\tau(x,y,z))$ has no ff-tree with depth $\le k$, hence $P\cup(\leftarrow A_i(x),\tau(x,y,z))$ has no ff-tree with depth $\le k+1$. Since $\tau\le\sigma$ and both τ and σ are ground, it follows that $\tau\equiv_x\sigma$. Hence $P\cup(\leftarrow A_i(x),\sigma(x))$ has no ff-tree with depth $\le k+1$.

So we proved that if $A_i(u)\in CT_P{\downarrow}k+1$, then $P\cup(\leftarrow A_i(x),\sigma)$ has no ff-tree with depth $\le k+1$. Suppose for all $1\le i\le n$, $P\cup(\leftarrow A_i(x),\sigma)$ has no ff-tree with $\le k+1$, then it follows by lemma 5.4 that $P\cup(\leftarrow A_1(x),\ldots,A_n(x),\sigma(x))$ has no such ff-tree. □

Theorem 5.13 is an immediate consequence of the corresponding theorem in [8]. Its proof is much simpler than the proof in [12] for SLD-resolution, due to the fact that the model gfp(CT_P) is a complete Herbrand model. In the case of SLD-resolution, a completeness theorem for negation as failure was proved by JAFFAR, LASSEZ & LLOYD [9] and WOLFRAM, MAHER & LASSEZ [15] by constructing a model for comp(P)$\cup\{\exists(A_1\wedge...\wedge A_n\wedge\sigma_0)\}$ which is not a Herbrand model. This serious complication in the proof is overcome by the fact that $CT_P\downarrow\omega$=gfp(CT_P) (for other interpretations of fixed points see for instance LEVI & PALAMIDESSI [11]).
We have the following corollary which cannot be obtained in ordinary SLD-semantics:

corollary 5.14 comp(P)$\cup\{A\wedge\sigma\}$ has a complete Herbrand model iff it has a model.

Acknowledgements
At this place I especially want to thank Krzysztof Apt for his clarifying comments, Jos Baeten for pointing out some errors in a draft version and Johan van Benthem for giving his support to this paper.

REFERENCES

[1] Apt, K.R. & van Emden, M.H., *Contributions to the Theory of Logic Programming*, JACM,29,3 (july 1982),841-862.

[2] Colmerauer, A., *Prolog and Infinite Trees*, Clark and Tarnlund (eds.), Logic Programming, Academic Press, New York, 1982, pp.231-251.

[3] Colmerauer, A., *Prolog II Reference Manual and Theoretical Model*, Groupe Intelligence Artificielle, Faculté des Sciences de Luming, Marseille, 1982.

[4] Courcelle, B., *Fundamental Properties of Infinite Trees*, Theoretical Computer Science, 25(2), pp.95-169, March 1983.

[5] van Emden, M.H. & Kowalski, R.A., *The Semantics of Predicate Logic as a Programming Language*, JACM,23,4 (October 1976),733-742.

[6] van Emden, M.H. & Lloyd, J.W., *A Logical Reconstruction of Prolog II*, Journal of Logic Programming, 1(2), pp.143-150, august 1984.

[7] Jaffar, J. & Lassez, J-L., *Constraint Logic Programming*, proc. POPL, pp. 111-119, Munich 1987.

[8] Jaffar, J., Lassez, J-L. & Maher, M.J., *Prolog II as an instance of the Logic Programming Language Scheme*, M. Wirsing (ed.), proc. IFIP conf., North Holland, 1987.

[9] Jaffar, J., Lassez, J-L. & Lloyd, J.W., *Completeness of the Negation as Finite Failure Rule*, IJCAI-83, Karlsruhe, 1983, 500-506.

[10] Jaffar, J. & Stuckey, P., *Semantics of Infinite Tree Logic Programming*, TCS, vol 46, pp. 141-158, 1986.

[11] Levi, G. & Palamidessi, C., *Contributions to the Semantics of Logic Perpetual Processes*, draft manuscript, Dipartimento di Informatica, Università di Pisa, Italy, 1987.

[12] Lloyd, J.W., *Foundations of Logic Programming*, Springer Verlag, Heidelberg, 1984.

[13] Maher, M.J., *Logic Semantics for a Class of Committed-choice Programs*, Proc. 4th. Int. Conf. on Logic Programming, Melbourne, pp. 858-876, 1987.

[14] Plaisted, D.A., *The Occur-Check Problem in Prolog*, Int. Symp. on Logic Programming, Atlantic City, 1984, 272-280.

[15] Wolfram, D.A., Maher, M.J. & Lassez, J-L., *A Unified Treatment of Resolution Strategies for Logic Programs*, Second International Logic Programming Conference, Uppsala, 1984, 263-276.