

Logic for Computer Science

Harald Ganzinger

Summer Term 2002

Part 2: Logic Programming

2

- Literature:**
1. Schöning: Logik für Informatiker, Kapitel 3.
 2. Hanus: Problemlösen mit Prolog, Teubner.
 3. Sterling, Shapiro: The Art of Prolog, MIT Press 86.
 4. Lloyd: Foundations of LP, Springer 84.
 5. **Apt: Logic Programming, in: Handbook of Theoretical Computer Science Vol. B, MIT Press/Elsevier 1990**

- History**
- Robinson 65: Resolution, Unifikation
 - Kowalski, Colmerauer ab 70: Prolog
 - D. Warren 75-79: efficient implementation techniques
 - Jaffar, Lassez 86: Constraint LP
 - Shapiro 85: Concurrent Prolog
 - Saraswat 90: Concurrent Constraint Programming
 - Languages: Prolog (II, III), CLP(X), AKL, Oz, Eclipse, Mozart

Syntax

4

A **logic program** P is a finite set of Horn clauses of the form

$$A_1 \wedge \dots \wedge A_n \rightarrow A, \quad n \geq 0$$

in FOL without equality. Alternative notations:

$$A \leftarrow A_1, \dots, A_n$$

$$A :- A_1, \dots, A_n.$$

A is called the **head**, A_1, \dots, A_n the **body** of the clause. A **query** (or **goal**) G is a (conjunctive, semantically) multiset $B_1 \wedge \dots \wedge B_k$ of atoms. As the negation of a query (for the purpose of refutational theorem proving) is a Horn clause

$$B_1 \wedge \dots \wedge B_k \rightarrow \perp,$$

$k \geq 0$, with **empty head**, queries are also written in the form

$$\leftarrow B_1, \dots, B_k$$

$$:- B_1, \dots, B_k.$$

$$\square, \text{ if } k = 0.$$

```
append([],L,L).
append([X|Ls],List0,[X|List1]):-
    append(Ls,List0,List1).
```

% Concatenation of all elements in a list of lists

```
append([], []).
append([L|Ls], List0) :-
    append(L, List1, List0),
    append(Ls, List1).
```

Here, [- | -] is mixfix-notation for the binary function symbol “-.-” (read “cons”) constructing binary trees. [] is the empty tree (the empty list).

[$A_1, \dots, A_{k-1}, A_k \mid X$] is an abbreviation for [$A_1 \mid \dots \mid [A_{k-1} \mid [A_k \mid X]] \dots$].

Variables begin with an uppercase letter or with an underscore “_”.

```
% perm(+List, ?Perm)
% is true when List and Perm are permutations of each other. The main
% use of perm/2 is to generate permutations. List must be
% a proper list. Perm may be partly instantiated.
```

```
perm([], []).
perm([X|Xs], Ys1) :-
    perm(Xs, Ys),
    insert(Ys, X, Ys1).
```

```
insert(L, X, [X|L]).
insert([H|T], X, [H|L]) :-
    insert(T, X, L).
```

If P is a logic program and G a query a substitution σ is called a **solution** of G , of

$$P \models G\sigma.$$

Again we assume that clauses (program clauses as well as the negation of the query) are implicitly universally quantified.

Hence the variables in the query G are **implicitly existentially quantified**, and solutions are explicit **witnesses** for the existence of terms with the required properties.

Computing solutions is a deductive service that goes beyond the mere refutation of a query. A solution is an **output** of the program given the query as **input**. There can be none, one, or many outputs to any given input.

Let P be a logic program. The **operational semantics** of P is given via a binary relation \vdash_P over pairs $(G; \sigma)$ of goals and substitutions, defined by this rule (to be applied modulo associativity and commutativity of “;” in goals):

$$(G, A; \sigma) \vdash_P ((G, A_1, \dots, A_n)\sigma'; \sigma' \circ \sigma)$$

if there exists a variant (with fresh variables not appearing in $\text{var}(G, A) \cup \text{dom}(\sigma) \cup \text{codom}(\sigma)$)

$$A_0 \leftarrow A_1, \dots, A_n$$

of a P -clause such that $\sigma' = \text{mgu}(A, A_0)$, $\text{codom}(\sigma') \cap \text{dom}(\sigma) = \emptyset$, $\text{codom}(\sigma') \subseteq \text{var}(A, A_0)$.^a A is called the **selected atom** of this computation step.

σ is called an **answer substitution** for G wrt. P whenever

$$(G; []) \vdash_P^* (\square; \sigma),$$

where $[]$ is the empty substitution (the identity) and \square the empty query.

^aIf σ' is computed after MM, the requirements about idempotence and co-domain of σ' are automatically satisfied.

Resolution for Definite clauses:

$$(G, A; \sigma) \vdash_P ((G, A_1, \dots, A_n)\sigma'; \sigma' \circ \sigma)$$

is a step of resolution (with collecting unifiers) between a program clause and a negated query:

$$\frac{A_1 \wedge \dots \wedge A_n \rightarrow A_0 \quad G \wedge A \rightarrow \perp}{(G \wedge A_1 \wedge \dots \wedge A_n \rightarrow \perp)\sigma'}$$

Linearity: one never resolves two program clauses \Rightarrow linear proof structure (why complete?)

Selection: A don't-care non-deterministically selected (e.g., by a selection function)

THEOREM 2.1 If two derivations $(G; \sigma) \vdash_{P^*} (G_1; \sigma_1)$ and $(G; \sigma) \vdash_{P^*} (G_2; \sigma_2)$ differ only in the variable names introduced when building variants of program clauses then G_1 and G_2 , and also σ_1 und σ_2 , respectively, are equal up to variable renaming.

The **SLD tree** for a query Q is a marked, possibly infinite tree where:

- Nodes are marked by pairs $(G; \sigma)$ of goals and substitutions.
- In each node with a non-empty goal, one goal atom is selected.
- A node marked $(G; \sigma)$ has a descendant $(G'; \sigma')$ if, and only if, $(G; \sigma) \vdash_P (G'; \sigma')$ by expanding the selected atom A in G with a don't-care non-deterministically chosen variant^a of a program clause whose head is unifiable with A .
- The root of the tree is marked by $(Q; [])$.
- A leaf of the tree with an empty goal is called a **success node**; its substitution part represents an answer substitution for Q .
- A leaf of the tree with a non-empty goal is called a **failure node**.

^aBecause of Theorem 2.1 we only want one descendant for each program clause whose head (after variable renaming) is unifiable with A !

Prolog has a specific (incomplete) operational semantics for building and traversing the \vdash_P -tree: First it is assumed that program clauses are ordered (in the way they are written down in a program text). Then the tree is constructed and traversed in the following manner:

- Always the first atom in any goal is selected.
- The SLD tree is traversed depth-first from left to right. That is, Prolog assumes that the descendants of a node are **ordered** in exactly the same way the program clauses that are applied to expand the node are ordered.
- Built-in predicates are evaluated only if all arguments are ground. A node is an **error node** if the selected atom has a built-in predicate but is not ground.
- Cuts (denoted by the built-in predicate “!”) cut off subtrees.

Consider a predicate $\text{config}(L, A, S, R)$, where

L : mirrored left half of the tape

A : symbol currently being read

S : current state

R : right half of the tape.

Use the constant $\#$ to denote the space symbol (for padding).

Translate the transition relation δ of the TM into these program clauses:

$$\text{config}(L, A, S, R) \quad :- \quad \text{config}(L, A', S', R). \\ \text{for } (S', A', n) \in \delta(S, A)$$

$$\text{config}(L, A, S, [B|R]) \quad :- \quad \text{config}([A'|L], B, S', R). \\ \text{config}(L, A, S, []) \quad :- \quad \text{config}([A'|L], \#, S', []). \\ \text{for } (S', A', r) \in \delta(S, A)$$

analogously for transitions of the form $(S', A', l) \in \delta(S, A)$

Add “facts” of the form

$$\text{config}(L, A, s_e, R).$$

for each $s_e \in S_e$ (= set of final states).

THEOREM 2.2 Let $M = (\Sigma, \Gamma, S, s_0, S_e, \delta)$ a Turing machine and let P_M the logic program constructed from M as described before. Then, for any word $\omega = a_1 a_2 \dots a_k, k \geq 0$, over the input alphabet Σ of M , we have that

$$\omega \in L(M) \Leftrightarrow (\text{config}([], a'_1, s_0, [a_2, \dots, a_k]); []) \vdash_{P_M}^* (\square; \sigma)$$

where

$$a'_1 = \begin{cases} a_1 & \text{if } k \geq 1 \\ \# & \text{otherwise} \end{cases}$$

- Parameters have no unique direction “in” or “out”

$$:- \text{rev}(L, [1, 2, 3]). \\ :- \text{rev}([1, 2, 3], L1). \\ :- \text{rev}([1, X], [2, Y]).$$

- Symbolic programming by structural induction

$$\text{rev}([], []). \\ \text{rev}([X|Xs], Ys) :- \dots$$

- Generate and test

$$\text{sort}(Xs, Ys) :- \text{perm}(Xs, Ys), \text{ordered}(Ys).$$

- Logic variables in open structures

$$\text{append_dl}(Xs/Ys, Ys/Zs, Xs/Zs).$$

$$\text{in_dict}(K, [(K, V)|D], V).$$

$$\text{in_dict}(K, [(K1, V1)|D], V) :- K \backslash== K1, \text{in_dict}(K, D, V).$$

- Context-free syntactic analysis

$$\% A ::= BC \quad ==> \\ 'A'(Xs/Ys) :- 'B'(Xs/Zs), 'C'(Zs/Ys). \\ \% A ::= a \quad ==> \\ 'A'([a|Xs]/Xs).$$

add more parameters for semantic analysis and code generation

General Assumption: Σ has infinitely many constants (otherwise technically more complicated).

Definition T_P -Operator: Let P be a L.P. and I a Herbrand interpretation over Σ .

$$T_P(I) = \{A\sigma \mid \text{there exists } A \leftarrow A_1, \dots, A_n \in P : \\ A\sigma \in G_\Sigma(A) \text{ and } \forall 1 \leq i \leq n : A_i\sigma \in I\}$$

Hence, $T_P : 2^{B_\Sigma} \rightarrow 2^{B_\Sigma}$, where B_Σ the set of Σ -ground atoms, called the **Herbrand base**.

T_P is called the **immediate consequence operator** associated with P .

Bottom-up view of P : Herbrand interpretations $I_n = T_P^n(\emptyset)$ and their limit $I_P = \bigcup_{n=0}^{\infty} I_n$

$$\begin{aligned} & \text{even}(0). \\ \text{even}(ssx) & \leftarrow \text{even}(x), \text{odd}(sx). \\ \text{odd}(sx) & \leftarrow \text{even}(x). \\ \text{odd}(x) & \leftarrow \text{even}(sx). \end{aligned}$$

$$\begin{aligned} I_0 &= \emptyset \\ I_1 &= \{\text{even}(0)\} \\ I_2 &= I_1 \cup \{\text{odd}(s0)\} \\ I_3 &= I_2 \cup \{\text{even}(ss0)\} \\ &\dots \\ I_n &= \{\text{even}(s^{2m}(0)) \mid 2m < n\} \cup \{\text{odd}(s^{2m+1}(0)) \mid 2m + 1 < n\} \\ &\vdots \\ I_P &= \bigcup_{n=0}^{\infty} I_n = \{\text{even}(s^{2m}(0)) \mid m \in \mathbb{N}\} \cup \{\text{odd}(s^{2m+1}(0)) \mid m \in \mathbb{N}\} \end{aligned}$$

$f : 2^U \rightarrow 2^U$ is called **monotone** if whenever $I \subseteq J \subseteq U$ then $f(I) \subseteq f(J)$.

$f : 2^U \rightarrow 2^U$ is called **continuous** if for all chains $I_0 \subseteq I_1 \subseteq \dots$ of subsets $I_j \subseteq U$ it holds that

$$f\left(\bigcup_{j=0}^{\infty} I_j\right) = \bigcup_{j=0}^{\infty} f(I_j)$$

PROPOSITION 2.1 (i) f continuous $\Rightarrow f$ monotone

(ii) f continuous $\not\Rightarrow f$ monotone

(iii) f monotone then for all chains $I_0 \subseteq I_1 \subseteq \dots$, with $I_j \subseteq U$:

$$f\left(\bigcup_{j=0}^{\infty} I_j\right) \supseteq \bigcup_{j=0}^{\infty} f(I_j)$$

NB: all these definitions also work for **cpo's** (complete partial orders) in general.

Tarski's Fixpoint Theorem

THEOREM 2.3 (TARSKI)

Let $O = \{I \subseteq U \mid F(I) \subseteq I\}$ be the set of all **pre-fixpoints** of a monotone function $F : 2^U \rightarrow 2^U$. Then

$$\mu F := \bigcap_{I \in O} I$$

is the least fixpoint of F .

Proof. (i) $\bigcap_{I \in O} I$ is **fixpoint**: To be shown: $F(\bigcap_{I \in O} I) = \bigcap_{I \in O} I$

$$I \supseteq F(I) \supseteq F(\bigcap_{I \in O} I), \text{ for all } I \in O \quad (2. \text{ ineq. by monotonicity of } F)$$

$$\Rightarrow \boxed{\bigcap_{I \in O} I \supseteq \bigcap_{I \in O} F(I) \supseteq F(\bigcap_{I \in O} I)}$$

$$\Rightarrow F(\bigcap_{I \in O} I) \supseteq F(F(\bigcap_{I \in O} I)) \quad (F \text{ monotone})$$

$$\Rightarrow F(\bigcap_{I \in O} I) \in O$$

$$\Rightarrow \boxed{F(\bigcap_{I \in O} I) \supseteq \bigcap_{I \in O} I}$$

Altogether we obtain: $F(\bigcap_{I \in O} I) = \bigcap_{I \in O} I$

(ii) μF is the **least fixpoint**: $F(J) = J \Rightarrow J \in O \Rightarrow \bigcap_{I \in O} I \subseteq J$.

□

THEOREM 2.4 (KLEENE)

$$F \text{ continuous} \Rightarrow \mu F = \bigcup_{i=0}^{\infty} F^i(\emptyset)$$

Proof. By monotonicity of F , the $F^i(\emptyset)$ form an ascending chain.

$$\begin{aligned} F\left(\bigcup_{i=0}^{\infty} F^i(\emptyset)\right) &= \bigcup_{i=0}^{\infty} F^{i+1}(\emptyset) && (F \text{ continuous}) \\ &= \bigcup_{i=1}^{\infty} F^i(\emptyset) = \bigcup_{i=1}^{\infty} F^i(\emptyset) \cup \emptyset = \bigcup_{i=1}^{\infty} F^i(\emptyset) \cup F^0(\emptyset) \\ &= \bigcup_{i=0}^{\infty} F^i(\emptyset) \end{aligned}$$

$F(J) = J \Rightarrow F^i(\emptyset) \subseteq F^i(J) = J$, for all i , hence $\bigcup_{i=0}^{\infty} F^i(\emptyset) \subseteq J \square$

Application to Logic Programs

THEOREM 2.5 (Continuity of T_P)

Let P be a L.P. Then $T_P : 2^{B\Sigma} \rightarrow 2^{B\Sigma}$ is continuous.

THEOREM 2.6

I is a Herbrand model of P if, and only if, I is closed under the immediate consequence operator, that is, $T_P(I) \subseteq I$

Proof. (Reminder: For Herbrand interpretations I , variable assignments and ground substitutions are the same thing.)

“ \Rightarrow ”: $A \in T_P(I) \Rightarrow A = B\sigma$, $B\sigma \leftarrow B_1\sigma, \dots, B_n\sigma \in G_\Sigma(P)$, and $B_i\sigma \in I$, for some program clause in P . As $I \models P$, also $A \in I$.

“ \Leftarrow ”: If $B_1\sigma, \dots, B_n\sigma \in I$ and $B \leftarrow B_1, \dots, B_n \in P$, we need to show that $B\sigma \in I$. This follows, however, as under the given assumptions $B\sigma \in T_P(I) \subseteq I$.

\square

Application to Logic Programs

THEOREM 2.7

Let $(I_k)_{k \in K}$ be a family of Herbrand models of P . Then $\bigcap_{k \in K} I_k$ is also a (Herbrand) model of P .

Proof. Let $B\sigma \leftarrow B_1\sigma, \dots, B_n\sigma \in G_\Sigma(P)$, such that $B_j\sigma \in \bigcap_{k \in K} I_k$, $1 \leq j \leq n$,
 $\Rightarrow B_j\sigma \in I_k$, $1 \leq j \leq n$, $k \in K$
 $\Rightarrow B\sigma \in I_k$, $k \in K$ [I_k model]
 $\Rightarrow B\sigma \in \bigcap I_k \square$

Convince yourself that it is the particular syntactic restrictions of logic programs that make this property become true.

The Canonical Model of a Program

THEOREM 2.8 $\mu T_P = \bigcap_{T_P(I) \subseteq I} I$ is the least fixpoint of T_P and the minimal Herbrand model of P .

Proof. Tarski's theorem in combination with the two preceding theorems.
 \square

$I_P := \mu T_P$ is called the **canonical model** of the program P .

THEOREM 2.9 1. $I_P = I_{G_\Sigma(P)} = \bigcup_{i=0}^{\infty} T_P^i(\emptyset) = \bigcap_{T_P(I) \subseteq I} I$

2. Let $G(\vec{x})$ be a query. Then $P \models \forall \vec{x} G \Leftrightarrow I_P \models \forall \vec{x} G$.

3. Let $G(\vec{x})$ be a query. Then $P \models \exists \vec{x} G \Leftrightarrow I_P \models \exists \vec{x} G$.

(I_P is canonical since validity in I_P coincides with what is entailed by P .)

Proof. We show “ \Leftarrow ” in 2; the other statements are then easy or follow directly from what has been shown already. We need an auxiliary definition, the **index of an atom**.

Let $ind : B_\Sigma \rightarrow \mathbb{N} \cup \{\infty\}$ such that

$$ind(A) = \min(\{n \in \mathbb{N} \mid A \in T_P^n(\emptyset)\} \cup \{\infty\}).$$

With this definition, the T_P -ordering $A \succ_P B :\Leftrightarrow ind(A) > ind(B)$ is noetherian.

For ground queries $G = A_1 \wedge \dots \wedge A_k$, $k \geq 0$, the index is defined as

$$ind(G) = \max(\{ind(A_j)\} \cup \{0\})$$

$$ind(G') = \infty \Rightarrow I_P \not\models G'$$

($ind(A)$ cannot be effectively computed as it might be ∞ . $ind(A)$ is a theoretical construction for the proof of canonicity of I_P .)

Let $I_P \models \forall \vec{x}G$. Then also $I_P \models G'$ for $G' = G\rho$, with $\rho : x \in var(G) \mapsto c_x \in \Omega$, where the c_x are pairwise different (Skolem-) constants that do not occur in P or G . We show that $P \models G'$.^a

Induction over $ind(G') = n < \infty$

a) $n = 0 \Rightarrow k = 0 \Rightarrow G' = \top \Rightarrow P \models G'$.

b) $n > 0$: to be shown: $P \models A_j\rho$, $1 \leq j \leq k$

b.1) $ind(A_j\rho) < n \Rightarrow P \models A_j\rho$ (ind.hyp.)

b.2) $ind(A_j\rho) = n \Rightarrow A_j\rho \in T_P^n(\emptyset) \setminus T_P^{n-1}(\emptyset)$

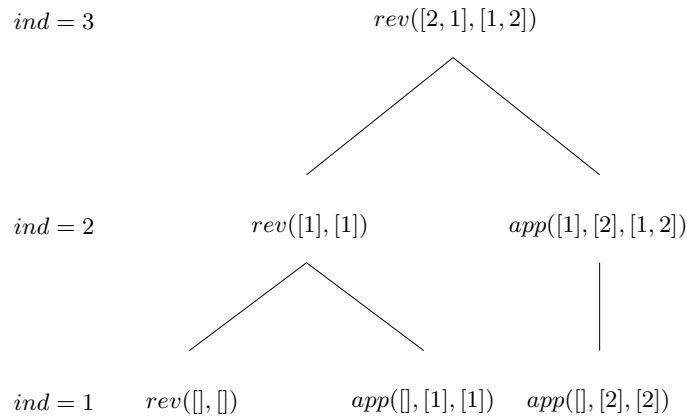
\Rightarrow there exist $B_1, \dots, B_m \rightarrow B$ in P and σ such that $B\sigma = A_j\rho$ and

$B_i\sigma \in T_P^{n-1}(\emptyset)$, in particular $ind(B_i\sigma) < n$.

$\Rightarrow P \models B_i\sigma$, $1 \leq i \leq m$ (ind.hyp.)

$\Rightarrow P \models B\sigma [= A_j\rho]$. \square

^aThe fact that $P \models G'$ implies $P \models \forall \vec{x}G$ is left as an exercise.



$ind(A) \leq n \Leftrightarrow$ there exists a direct proof (cf. below) of depth n .

Let \succ be a well-founded and total extension of the T_P -ordering \succ_P . We will show that the (sub-)set of **reductive instances** of P

$$P_{red} = \{A_1, \dots, A_k \rightarrow A \in G_\Sigma(P) \mid A \succ A_i, 1 \leq i \leq k\}$$

have the same minimal model than the entire set of clauses P .

THEOREM 2.10

(i) $P \subseteq Q \Rightarrow I_P \subseteq I_Q$

(ii) $I_P = I_{G_\Sigma(P)}$

(iii) $I_P = I_{P_{red}}$

Proof. (i), (ii) are easy (exercise).

We abbreviate $G_\Sigma(P)$ by GP and show that $T_{GP}^n(\emptyset) = T_{P_{red}}^n(\emptyset)$, for all n , by induction.

$n = 0$: $T_{GP}^0(\emptyset) = \emptyset = T_{P_{red}}^0(\emptyset)$

$n > 0$: $T_{GP}^n(\emptyset) \supseteq T_{P_{red}}^n(\emptyset)$ is true since $GP \supseteq P_{red}$ and $T_{GP}^{n-1}(\emptyset) = T_{P_{red}}^{n-1}(\emptyset)$ by ind.hyp. We now prove the reverse direction. Let $A \in T_{GP}^n(\emptyset)$. If $A \in T_{GP}^{n-1}(\emptyset)$, then, by ind.hyp., also $A \in T_{P_{red}}^{n-1}(\emptyset) \subseteq T_{P_{red}}^n(\emptyset)$. If $A \in T_{GP}^n(\emptyset) \setminus T_{GP}^{n-1}(\emptyset)$, then there exists a clause $C = A \leftarrow A_1, \dots, A_k$ in GP such that $A_i \in T_{GP}^{n-1}(\emptyset)$. Therefore, $ind(A) = n$ and $ind(A_i) < n$ and, thus, $C \in P_{red}$. By ind.hyp. we also have $A_i \in T_{P_{red}}^{n-1}(\emptyset)$, therefore, $A \in T_{P_{red}}^n(\emptyset)$.

□

I_P is a Special Case of our Model Construction

Let again \succ be a well-founded and total extension of the T_P -ordering \succ_P . Let $I_{G_\Sigma(P)}^\succ$ the candidate model wrt. \succ (construction for the case without selection) for $G_\Sigma(P)$ (cf. ch. 1).

THEOREM 2.11

$$I_P = I_{G_\Sigma(P)}^\succ = I_{P_{red}}^\succ$$

Proof.

$$\begin{aligned} I_P &= I_{P_{red}} && \text{by Theorem 2.10} \\ &\subseteq I_{P_{red}}^\succ && P_{red} \text{ is saturated wrt. } Res^\succ, \text{ hence, } I_{P_{red}}^\succ \models P_{red} \\ &&& \text{and since } I_{P_{red}} \text{ minimal (cf. Theorem 2.8)} \\ &= I_{G_\Sigma(P)}^\succ && \text{only reductive clauses are productive} \\ &= I_{P_{red}} && \text{candidate models are minimal} \\ &= I_P && \text{by Theorem 2.10} \end{aligned}$$

□

2.2 Properties of SLD-Resolution

Correctness and Answer Completeness of SLD-Resolution

Let S be a **LP-selection function**, i.e., a selection function selecting in each goal G (that is, negative clause $\neg G$) one literal, but selecting nothing in any program clause.

Let again \succ denote a well-founded total extension of the T_P -ordering.

By $\vdash_{P,S}$ we denote the SLD-resolution derivation relation \vdash_P whereby in each goal one expands the selected (according to S) atom.

THEOREM 2.12 For arbitrary queries $G(\vec{x})$ and substitutions σ with $dom(\sigma) \subseteq var(G)$ it holds that:

$$P \models G\sigma \iff (G; \square) \vdash_{P,S}^* (\square; \tau), \text{ for some } \tau \text{ with } \tau|_{var(G)} \leq \sigma.$$

Correctness is easy; let's deal with the completeness part (“ \Rightarrow ”):

Let $var(G) = \{x_1, \dots, x_n\}$. We apply an “answer predicate trick” and introduce a new n -ary predicate α . (α simulates the substitution part in SLD-derivations.)

Let ρ be a substitution mapping any variable x in $codom(\sigma) \cup (var(G) \setminus dom(\sigma))$ to a unique constant c_x (cf. proof of Theorem 2.9).

Let C_α denote the unit clause $\alpha(x_1, \dots, x_k)\sigma\rho$ and G_α the (extended) query $G, \alpha(x_1, \dots, x_k) \rightarrow$. Now consider any selection function S_α selecting in each clause not containing α according to S , and in each query of the form $H, \alpha(\dots) \rightarrow$, with α not in H , what is selected by S in $H \rightarrow$. (In other words, the α -atom can only be selected if no other atom appears in a query.)

We now show that refutational completeness of ordered resolution $Res_{S_\alpha}^>$ with selection according to S_α (applied to the clause set $P^\alpha = P \cup \{C_\alpha\}$) implies that

$$(G, \alpha(x_1, \dots, x_k); \square) \vdash_{P^\alpha; S_\alpha}^* (\alpha(x_1 \dots, x_n)\tau; \tau) \vdash_{P^\alpha; S_\alpha} (\square; \tau)$$

and, thus, $(G; \square) \vdash_{P; S}^* (\square; \tau)$, for some $\tau \leq \sigma$.

$$\begin{aligned} P \models G\sigma &\Leftrightarrow P^\alpha \models G_\alpha\sigma\rho \\ &\Leftrightarrow I_{P^\alpha} \models G_\alpha\sigma\rho && \text{[Theorem 2.9]} \\ &\Leftrightarrow I_{P_{red}^\alpha} \models G_\alpha\sigma\rho && \text{[Theorem 2.11]} \\ &\Leftrightarrow P_{red}^\alpha \models G_\alpha\sigma\rho && \text{[Theorem 2.9]} \\ &\Leftrightarrow P_{red}^\alpha \cup \{\neg G_\alpha\sigma\rho\} \models \perp \\ &\Leftrightarrow \square \in (Res_{S_\alpha}^>)^*(P_{red}^\alpha \cup \{\neg G_\alpha\sigma\rho\}) && \text{[compl. of ord. resol.]} \\ &\Leftrightarrow (G_\alpha\sigma\rho; \square) \vdash_{P_{red}^\alpha; S_\alpha}^* (\square; \square) && \text{[resolution inferences]} \\ &\quad \text{between two premises both in } P_{red}^\alpha \text{ are not possible} \\ &\Rightarrow (G_\alpha; \square) \vdash_{P; S_\alpha}^* (\alpha(x_1 \dots, x_n)\tau; \tau) && \text{[Lifting Lemma]} \\ &\quad \vdash_{\{C_\alpha\}; S_\alpha} (\square; \tau) \end{aligned}$$

As $C_\alpha = \alpha(x_1, \dots, x_k)\sigma\rho$, the last step in \vdash is possible only if $\tau|_{var(G)} \leq \sigma$. Because of $(G_\alpha; \square) \vdash_{P; S_\alpha}^* (\alpha(x_1 \dots, x_n)\tau; \tau)$ one also has $(G; \square) \vdash_{P; S}^* (\square; \tau)$.

A (direct) proof of an atom A (wrt. P) is an unordered, marked (by atoms, not necessarily ground) tree Π such that

- (i) the root of Π is marked by A ,
- (ii) for each node v in Π marked B

there exists a clause $B_0 \leftarrow B_1, \dots, B_n$ in P such that $B = B_0\sigma$, and v has exactly $n \geq 0$ descendants and these are marked $B_1\sigma, \dots, B_n\sigma$.

A is called (directly) provable from P , written $P \Vdash A$, iff there exists a proof of A from P .

THEOREM 2.13 Let A be an atom and σ a substitution.

$$P \models A\sigma \Leftrightarrow P \Vdash A\sigma$$

Proof: exercise.

- canonical model: $P \models Q\vec{x} G(\vec{x}) \Leftrightarrow I_P \models Q\vec{x} G(\vec{x})$
- minimal model [= intersection of all models]

$$I_P = \mu T_P = \bigcap_{T_P(I) \subseteq I} I$$

- recursively enumerable semantics: $I_P = \mu T_P = \bigcup_{i=0}^{\infty} T_P^i(\emptyset)$
- SLD-resolution:
 - selection as before (but we don't select in program clauses)
 - linearity (no inferences between any two program clauses in P)
 - \Rightarrow SLD-proofs can be represented as paths in SLD-trees

$$G \vdash_{P; S} G' \vdash_{P; S} \dots \vdash_{P; S} \square$$

- answer completeness: the set of substitutions τ with

$$(G; \square) \vdash_{P; S}^* (\square; \tau)$$

- subsumes all answer substitutions for G ; completeness via $Res_{S_\alpha}^>$ with \succ based on the T_P -ordering.
- direct proofs give us alternate tree-like proof structures