

Chapter 9

SLD-Resolution And Logic Programming (PROLOG)

9.1 Introduction

We have seen in Chapter 8 that the resolution method is a complete procedure for showing unsatisfiability. However, finding refutations by resolution can be a very expensive process in the general case. If subclasses of formulae are considered, more efficient procedures for producing resolution refutations can be found. This is the case for the class of Horn clauses. A Horn clause is a disjunction of literals containing at most one positive literal. For sets of Horn clauses, there is a variant of resolution called SLD-resolution, which enjoys many nice properties. SLD-resolution is a special case of a refinement of the resolution method due to Kowalski and Kuehner known as SL-resolution (Kowalski and Kuehner, 1970), a variant of Model Elimination (Loveland, 1978), and applies to special kinds of Horn clauses called definite clauses. We shall present SLD-resolution and show its completeness for Horn clauses.

SLD-resolution is also interesting because it is the main computation procedure used in PROLOG. PROLOG is a programming language based on logic, in which a computation is in fact a refutation. The idea to define a program as a logic formula and view a refutation as a computation is a very fruitful one, because it reduces the complexity of proving the correctness of programs. In fact, it is often claimed that logic programs are obviously correct, because these programs “express” the assertions that they should satisfy. However, this is not quite so, because the notion of correctness is relative, and

one still needs to define the semantics of logic programs in some independent fashion. This will be done in Subsection 9.5.4, using a model-theoretic semantics. Then, the correctness of SLD-resolution (as a computation procedure) with respect to the model-theoretic semantics will be proved.

In this chapter, as in Chapter 8, we begin by studying SLD-resolution in the propositional case, and then use the lifting technique to extend the results obtained in the propositional case to the first-order case. Fortunately, the lifting process goes very smoothly.

As in Chapter 4, in order to prove the completeness of SLD-resolution for propositional Horn clauses, we first show that Horn clauses have $GCNF'$ -proofs of a certain kind, that we shall call $GCNF'$ -proofs in SLD-form. Then, we show that every $GCNF'$ -proof in SLD-form can be mapped into a linear SLD-refutation. Hence, the completeness proof for SLD-resolution is constructive.

The arguments used for showing that every unsatisfiable Horn clause has a $GCNF'$ -proof in SLD-form are quite basic and combinatorial in nature. Once again, the central concept is that of *proof transformation*.

We conclude this chapter by discussing the notion of logic program and the idea of viewing SLD-resolution as a computation procedure. We provide a rigorous semantics for logic programs, and show the correctness and completeness of SLD-resolution with respect to this semantics.

The contents of Section 9.5 can be viewed as the theoretical foundations of logic programming, and PROLOG in particular.

9.2 $GCNF'$ -Proofs in SLD-Form

First, we shall prove that every unsatisfiable propositional Horn clause has a $GCNF'$ -proof of a certain type, called a proof in SLD-form. In order to explain the method for converting a $GCNF'$ -proof into an SLD-resolution proof, it is convenient to consider the special case of sets of Horn clauses, containing exactly one clause containing no positive literals (clause of the form $\{\neg P_1, \dots, \neg P_m\}$). Other Horn clauses will be called *definite clauses*.

9.2.1 The Case of Definite Clauses

These concepts are defined as follows.

Definition 9.2.1 A *Horn clause* is a disjunction of literals containing at most one positive literal. A Horn clause is a *definite clause* iff it contains a (single) positive literal. Hence, a definite clause is either of the form

$$\{Q\}, \quad \text{or} \quad \{\neg P_1, \dots, \neg P_m, Q\}.$$

A Horn clause of the form

$$\{\neg P_1, \dots, \neg P_m\}$$

is called a *negative clause* or *goal clause*.

For simplicity of notation, a clause $\{Q\}$ will also be denoted by Q . In the rest of this section, we restrict our attention to sets S of clauses consisting of definite clauses except for one goal clause. Our goal is to show that for a set S consisting of definite clauses and of a single goal B , if S is *GCNF'*-provable, then there is a proof having the property that whenever a $\vee : left$ rule is applied to a definite clause $\{\neg P_1, \dots, \neg P_m, Q\}$, the rule splits it into $\{\neg P_1, \dots, \neg P_m\}$ and $\{Q\}$, the sequent containing $\{Q\}$ is an axiom, and the sequent containing $\{\neg P_1, \dots, \neg P_m\}$ does not contain $\neg Q$.

EXAMPLE 9.2.1

Consider the set S of Horn clauses with goal $\{\neg P_1, \neg P_2\}$ given by:

$$S = \{\{P_3\}, \{P_4\}, \{\neg P_1, \neg P_2\}, \{\neg P_3, \neg P_4, P_1\}, \{\neg P_3, P_2\}\}.$$

The following is a *GCNF'*-proof:

$$\frac{\frac{P_3, \neg P_3 \rightarrow P_4, \neg P_4 \rightarrow \neg P_1, P_1 \rightarrow P_3, \neg P_2, \{\neg P_3, P_2\} \rightarrow}{P_3, P_4, \{\neg P_3, \neg P_4\} \rightarrow P_3, \{\neg P_1, \neg P_2\}, P_1, \{\neg P_3, P_2\} \rightarrow}}{\frac{\neg P_2, P_2 \rightarrow P_3, \neg P_3 \rightarrow}{P_3, P_4, \{\neg P_1, \neg P_2\}, \{\neg P_3, \neg P_4, P_1\}, \{\neg P_3, P_2\} \rightarrow}}$$

Another proof having the properties mentioned above is

$$\frac{\frac{\frac{P_3, \neg P_3 \rightarrow P_4, \neg P_4 \rightarrow \neg P_1, P_1 \rightarrow P_3, P_4, \{\neg P_3, \neg P_4\} \rightarrow \neg P_2, P_2 \rightarrow P_3, \neg P_3 \rightarrow}{P_3, P_4, \neg P_1, \{\neg P_3, \neg P_4, P_1\} \rightarrow P_3, \neg P_2, \{\neg P_3, P_2\} \rightarrow}}{\frac{P_3, \neg P_3 \rightarrow P_4, \neg P_4 \rightarrow}{P_3, P_4, \{\neg P_1, \neg P_2\}, \{\neg P_3, \neg P_4, P_1\}, \{\neg P_3, P_2\} \rightarrow}}}$$

Observe that in the above proof, the $\vee : left$ rule is first applied to the goal clause $\{\neg P_1, \neg P_2\}$, and then it is applied to split each definite clause $\{\neg Q_1, \dots, \neg Q_m, Q\}$ into $\{\neg Q_1, \dots, \neg Q_m\}$ and $\{Q\}$, in such a way that the sequent containing $\{Q\}$ is the axiom $Q, \neg Q \rightarrow$. Note also that each clause $\{\neg Q_1, \dots, \neg Q_m\}$ resulting from splitting a definite clause as indicated above is the only goal clause in the sequent containing it.

9.2.2 *GCNF'*-Proofs in SLD-Form

The above example suggests that if a set of definite clauses with goal B is *GCNF'*-provable, it has a proof obtained by following rules described below, starting with a one-node tree containing the goal $B = \{\neg P_1, \dots, \neg P_m\}$:

(1) If no leaf of the tree obtained so far contains a clause consisting of a single negative literal $\neg Q$ then,

As long as the tree is not a *GCNF'*-proof tree, apply the $\vee : left$ rule to each goal clause B of the form $\{\neg Q_1, \dots, \neg Q_m\}$ ($m > 1$) in order to form m immediate descendants of B , else

(2) For every goal clause consisting of a single negative literal $\neg Q$, find a definite clause $\{\neg P_1, \dots, \neg P_k, Q\}$ (or Q when $k = 0$), and split $\{\neg P_1, \dots, \neg P_k, Q\}$ using the $\vee : left$ rule in order to get the axiom $\neg Q, Q \rightarrow$ in one node, $\{\neg P_1, \dots, \neg P_m\}$ in the other, and drop $\neg Q$ from that second node.

Go back to (1).

It is not clear that such a method works, and that in step (2), the existence of a definite clause $\{\neg P_1, \dots, \neg P_k, Q\}$ such that Q cancels $\neg Q$ is guaranteed. However, we are going to prove that this is always the case. First, we define the type of proofs arising in the procedure described above.

Definition 9.2.2 Given a set S of clauses consisting of definite clauses and of a single goal B , a *GCNF'*-proof is in *SLD-form* iff the conditions below are satisfied:

For every node B in the tree that is not an axiom:

(1) If the set of clauses labeling that node does not contain any clause consisting of a single negative literal $\neg Q$, then it contains a single goal clause of the form $\{\neg Q_1, \dots, \neg Q_m\}$ ($m > 1$), and the $\vee : left$ rule is applied to this goal clause in order to form m immediate descendants of B .

(2) If the set of clauses labeling that node contains some single negative literal, for such a clause $\neg Q$, there is some definite clause

$$\{\neg P_1, \dots, \neg P_k, Q\},$$

($k > 0$), such that the $\vee : left$ rule is applied to

$$\{\neg P_1, \dots, \neg P_k, Q\}$$

in order to get the axiom $\neg Q, Q \rightarrow$ and a sequent containing the single goal clause $\{\neg P_1, \dots, \neg P_k\}$.

We are now going to prove that if a set of clauses consisting of definite clauses and of a single goal clause is provable in *GCNF'*, then it has a proof in SLD-form. For this, we are going to perform proof transformations, and use simple combinatorial properties.

9.2.3 Completeness of Proofs in SLD-Form

First, we need to show that every $GCNF'$ -provable set of clauses has a proof in which no weakenings takes place. This is defined as follows.

Definition 9.2.3 A $GCNF'$ -proof is *without weakenings* iff every application of the $\vee : left$ rule is of the form:

$$\frac{\Gamma, A_1, \dots, A_m \rightarrow \quad \Gamma, B \rightarrow}{\Gamma, (A_1 \vee B), \dots, (A_m \vee B) \rightarrow}$$

We have the following normal form lemma.

Lemma 9.2.1 If a set S of clauses is $GCNF'$ -provable, then a $GCNF'$ -proof without weakenings and in which all the axioms contain only literals can be constructed.

Proof: Since G' is complete, $S \rightarrow$ has a G' -proof T . By lemma 6.3.1 restricted to propositions, $S \rightarrow$ has a G' -proof T' in which all axioms are atomic. Using lemma 4.2.2, $S \rightarrow$ has a G' -proof T'' in which all axioms are atomic, and in which all applications of the $\vee : left$ rule precede all applications of the $\neg : left$ rule. The tree obtained from T'' by retaining the portion of the proof tree that does not contain $\neg : left$ inferences is the desired $GCNF'$ -proof. \square

The following permutation lemma is the key to the conversion to SLD-form.

Lemma 9.2.2 Let S be a set of clauses that has a $GCNF'$ -proof T . Then, for any clause C in S having more than one literal, for any partition of the literals in C into two disjunctions A and B such $C = (A \vee B)$, there is a $GCNF'$ -proof T' in which the $\vee : left$ rule is applied to $(A \vee B)$ at the root. Furthermore, if the proof T of S is without weakenings and all axioms contain only literals, the proof T' has the same depth as T .

Proof: Observe that representing disjunctions of literals as unordered sets of literals is really a convenience afforded by the associativity, commutativity and idempotence of \vee , but that this convenience does not affect the completeness of G' . Hence, no matter how C is split into a disjunction $(A \vee B)$, the sequent $\Gamma, (A \vee B) \rightarrow$ is G' -provable. By converting a G' -proof of $\Gamma, (A \vee B) \rightarrow$ given by lemma 9.2.1 into a $GCNF'$ -proof, we obtain a $GCNF'$ -proof without weakenings, and in which the $\vee : left$ rule is applied to A and B only after it is applied to $(A \vee B)$. If the $\vee : left$ rule applied at the root does not apply to $(A \vee B)$, it must apply to some other disjunction $(C \vee D)$. Such a proof T must be of the following form:

Tree T

$$\frac{\Pi_1 \quad \Pi_2}{\Gamma, (A \vee B), (C \vee D) \rightarrow}$$

where Π_1 is the tree

$$\frac{\frac{T_1}{\Gamma_1, A \rightarrow} \quad \frac{S_1}{\Gamma_1, B \rightarrow}}{\Gamma_1, (A \vee B) \rightarrow} \quad \frac{\frac{T_m}{\Gamma_m, A \rightarrow} \quad \frac{S_m}{\Gamma_m, B \rightarrow}}{\Gamma_m, (A \vee B) \rightarrow}$$

 R

$$\Gamma, (A \vee B), C \rightarrow$$

and where Π_2 is the tree

$$\frac{\frac{T'_1}{\Delta_1, A \rightarrow} \quad \frac{S'_1}{\Delta_1, B \rightarrow}}{\Delta_1, (A \vee B) \rightarrow} \quad \frac{\frac{T'_n}{\Delta_n, A \rightarrow} \quad \frac{S'_n}{\Delta_n, B \rightarrow}}{\Delta_n, (A \vee B) \rightarrow}$$

 S

$$\Gamma, (A \vee B), D \rightarrow$$

In the above proof, we have indicated the nodes to which the $\vee : left$ rule is applied, nodes that must exist since all axioms consist of literals. The inferences above $\Gamma, (A \vee B), C$ and below applications of the $\vee : left$ rule to $(A \vee B)$ are denoted by R , and the similar inferences above $\Gamma, (A \vee B), D$ are denoted by S . We can transform T into T' by applying the $\vee : left$ rule at the root as shown below:

Tree T'

$$\frac{\Pi'_1 \quad \Pi'_2}{\Gamma, (A \vee B), (C \vee D) \rightarrow}$$

where Π'_1 is the tree

$$\frac{T_1}{\Gamma_1, A \rightarrow} \quad \frac{T_m}{\Gamma_m, A \rightarrow} \quad \frac{T'_1}{\Delta_1, A \rightarrow} \quad \frac{T'_n}{\Delta_n, A \rightarrow}$$

$R \qquad \qquad \qquad S$

$$\frac{\Gamma, A, C \rightarrow \quad \Gamma, A, D \rightarrow}{\Gamma, A, (C \vee D) \rightarrow}$$

and where Π'_2 is the tree

$$\frac{S_1}{\Gamma_1, B \rightarrow} \quad \frac{S_m}{\Gamma_m, B \rightarrow} \quad \frac{S'_1}{\Delta_1, B \rightarrow} \quad \frac{S'_n}{\Delta_n, B \rightarrow}$$

$R \qquad \qquad \qquad S$

$$\frac{\Gamma, B, C \rightarrow \quad \Gamma, B, D \rightarrow}{\Gamma, B, (C \vee D) \rightarrow}$$

Clearly, $\text{depth}(T') = \text{depth}(T)$. \square

Note that T' is obtained from T by permutation of inferences. We need another crucial combinatorial property shown in the following lemma.

Lemma 9.2.3 Let S be an arbitrary set of clauses such that the subset of clauses containing more than one literal is the nonempty set $\{C_1, \dots, C_n\}$ and the subset consisting of the one-literal clauses is J . Assume that S is $GCNF'$ -provable, and that we have a proof T without weakenings such that all axioms consist of literals. Then, every axiom is labeled with a set of literals of the form $\{L_1, \dots, L_n\} \cup J$, where each literal L_i is in C_i , $i = 1, \dots, n$.

Proof: We proceed by induction on proof trees. Since S contains at least one clause with at least two literals and the axioms only contain literals, $\text{depth}(T) \geq 1$. If T has depth 1, then there is exactly one application of the \vee : rule and the proof is of the following form:

$$\frac{J, L_1 \rightarrow \quad J, L_2 \rightarrow}{J, (L_1 \vee L_2) \rightarrow}$$

Clearly, the lemma holds.

If T is a tree of depth $k + 1$, it is of the following form,

$$\frac{\frac{T_1}{\Gamma, A \rightarrow} \quad \frac{T_2}{\Gamma, B \rightarrow}}{\Gamma, (A \vee B) \rightarrow}$$

where we can assume without loss of generality that $C_n = (A \vee B)$. By the induction hypothesis, each axiom of T_1 is labeled with a set of clauses of the form $\{L_1, \dots, L_n\} \cup J$, where each literal L_i is in C_i for $i = 1, \dots, n - 1$, and either $L_n = A$ if A consists of a single literal, or L_n belongs to A . Similarly, each axiom of T_2 is labeled with a set of clauses of the form $\{L_1, \dots, L_n\} \cup J$, where each literal L_i is in C_i for $i = 1, \dots, n - 1$, and either $L_n = B$ if B consists of a single literal, or L_n belongs to B . Since the union of A and B is C_n , every axiom of T is labeled with a set of clauses of the form $\{L_1, \dots, L_n\} \cup J$, where each literal L_i is in C_i , $i = 1, \dots, n$. Hence, the lemma holds. \square

As a consequence, we obtain the following useful corollary.

Lemma 9.2.4 Let S be a set of Horn clauses. If S is GCNF'-provable, then S contains at least one clause consisting of a single positive literal, and at least one goal (negative) clause.

Proof: If S is an axiom, this is obvious. Otherwise, by lemma 9.2.3, if S is GCNF'-provable, then it has a proof T without weakenings such that every axiom is labeled with a set of literals of the form $\{L_1, \dots, L_n\} \cup J$, where each literal L_i is in C_i , $i = 1, \dots, n$, and J is the set of clauses in S consisting of a single literal. If J does not contain any positive literals, since every Horn clause C_i contains a negative literal say $\neg A_i$, the set $\{\neg A_1, \dots, \neg A_n\} \cup J$ contains only negative literals, and so cannot be an axiom. If every clause in J is positive and every clause C_i contains some positive literal say A_i , then $\{A_1, \dots, A_n\} \cup J$ contains only positive literals and cannot be an axiom. \square

In order to prove the main theorem of this section, we will need to show that the provability of a set of Horn clauses with several goals (negative clauses) reduces to the case of a set of Horn clauses with a single goal.

Lemma 9.2.5 Let S be a set of Horn clauses consisting of a set J of single positive literals, goal clauses N_1, \dots, N_k , and definite clauses C_1, \dots, C_m containing at least two literals.

If S is GCNF'-provable, then there is some i , $1 \leq i \leq k$, such that

$$J \cup \{C_1, \dots, C_m\} \cup \{N_i\}$$

is GCNF'-provable. Furthermore, if T is a GCNF'-proof of S without weakenings and such that the axioms contain only literals, $J \cup \{C_1, \dots, C_m\} \cup \{N_i\}$ has a proof of depth less than or equal to the depth of T .

Proof: We proceed by induction on proof trees. Let T be a $GCNF'$ -proof of S without weakenings and such that all axioms contain only literals.

Case 1: $J \cup \{C_1, \dots, C_m\} \cup \{N_1, \dots, N_k\}$ is an axiom. Then, one of the positive literals in J must be the conjugate of some negative clause N_i , and the lemma holds.

Case 2: The bottom $\vee : left$ rule is applied to one of the N_i . Without loss of generality, we can assume that it is $N_1 = \{\neg Q_1, \dots, \neg Q_j, \neg P\}$.

Letting $\mathcal{C} = C_1, \dots, C_m$, the proof is of the form

$$\frac{\frac{T_1}{J, \mathcal{C}, N_2, \dots, N_k, \{\neg Q_1, \dots, \neg Q_j\} \rightarrow} \quad \frac{T_2}{J, \mathcal{C}, N_2, \dots, N_k, \neg P \rightarrow}}{J, \mathcal{C}, N_1, \dots, N_k \rightarrow}$$

Observe that the bottom sequents of T_1 and T_2 satisfy the conditions of the induction hypothesis. There are two subcases. If both

$$J, C_1, \dots, C_m, \{\neg Q_1, \dots, \neg Q_j\} \rightarrow \quad \text{and} \\ J, C_1, \dots, C_m, \neg P \rightarrow$$

are provable, then

$$J, C_1, \dots, C_m, \{\neg Q_1, \dots, \neg Q_j, \neg P\} \rightarrow$$

is provable by application of the $\vee : rule$, and the lemma holds. If

$$J, C_1, \dots, C_m, N_i \rightarrow$$

is provable for some i , $2 \leq i \leq k$, then the lemma also holds.

Case 3: The bottom $\vee : rule$ is applied to one of the C_i . Without loss of generality, we can assume that it is $C_1 = \{\neg Q_1, \dots, \neg Q_j, P\}$. There are two subcases:

Case 3.1: Letting $\mathcal{N} = N_1, \dots, N_k$, the proof is of the form

$$\frac{\frac{T_1}{J, C_2, \dots, C_m, \mathcal{N}, \{\neg Q_1, \dots, \neg Q_j\} \rightarrow} \quad \frac{T_2}{J, P, C_2, \dots, C_m, \mathcal{N} \rightarrow}}{J, C_1, \dots, C_m, \mathcal{N} \rightarrow}$$

Again the induction hypothesis applies to both T_1 and T_2 . If

$$J, C_2, \dots, C_m, \{\neg Q_1, \dots, \neg Q_j\} \rightarrow \quad \text{is provable and} \\ J, P, C_2, \dots, C_m, \mathcal{N} \rightarrow \quad \text{is provable}$$

for some i , $1 \leq i \leq k$, then by the $\vee : rule$,

$$J, C_1, \dots, C_m, N_i \rightarrow$$

is also provable, and the lemma holds. If

$$J, C_2, \dots, C_m, N_i \rightarrow$$

is provable for some i , $1 \leq i \leq k$, then

$$J, C_1, \dots, C_m, N_i \rightarrow$$

is also provable (using weakening in the last $\vee : rule$).

Case 3.2: Letting $\mathcal{N} = N_1, \dots, N_k$, the proof is of the form

$$\frac{\frac{T_1}{J, C_2, \dots, C_m, \mathcal{N}, \{\neg Q_2, \dots, \neg Q_j, P\} \rightarrow} \quad \frac{T_2}{J, C_2, \dots, C_m, \neg Q_1, \mathcal{N} \rightarrow}}{J, C_1, \dots, C_m, \mathcal{N} \rightarrow}$$

Applying the induction hypothesis, either

$$J, C_2, \dots, C_m, N_i, \{\neg Q_2, \dots, \neg Q_j, P\}$$

is provable for some i , $1 \leq i \leq k$, and

$$J, C_2, \dots, C_m, \neg Q_1 \rightarrow$$

is provable, and by the $\vee : rule$, J, C_1, \dots, C_m, N_i is provable and the lemma holds. Otherwise,

$$J, C_2, \dots, C_m, N_i$$

is provable for some i , $1 \leq i \leq k$, and so J, C_1, \dots, C_m, N_i is also provable using weakening in the last $\vee : rule$. This concludes the proof. \square

We are now ready to prove the main theorem of this section.

Theorem 9.2.1 (Completeness of proofs in SLD-form) If a set S consisting of definite clauses and of a single goal $B = \{\neg P_1, \dots, \neg P_n\}$ is $GCNF'$ -provable, then it has a $GCNF'$ -proof in SLD-form.

Proof: Assume that S is not an axiom. By lemma 9.2.1, there is a $GCNF'$ -proof T without weakenings, and such that all axioms consist of literals. We proceed by induction on the depth of proof trees. If $depth(T) = 1$, the proof is already in SLD-form (this is the base case of lemma 9.2.3). If $depth(T) > 1$, by n applications of lemma 9.2.2, we obtain a proof tree T' having the same depth as T , such that the i -th inference using the $\vee : left$ rule is applied to $\{\neg P_i, \dots, \neg P_n\}$. Hence, letting $\mathcal{C} = C_1, \dots, C_m$, the tree T' is of the form:

$$\frac{\frac{T_{n-1}}{J, \mathcal{C}, \neg P_{n-1} \rightarrow} \quad \frac{T_n}{J, \mathcal{C}, \neg P_n \rightarrow}}{J, \mathcal{C}, \{\neg P_{n-1}, \neg P_n\} \rightarrow}$$

...

$$\frac{\frac{T_1}{J, \mathcal{C}, \neg P_1 \rightarrow} \quad \frac{\frac{T_2}{J, \mathcal{C}, \neg P_2 \rightarrow} \quad J, \mathcal{C}, \{\neg P_3, \dots, \neg P_n\} \rightarrow}{J, \mathcal{C}, \{\neg P_2, \dots, \neg P_n\} \rightarrow}}{J, \mathcal{C}, \{\neg P_1, \dots, \neg P_n\} \rightarrow}$$

where J is the set of clauses consisting of a single positive literal, and each clause C_i has more than one literal. For every subproof rooted with $J, C_1, \dots, C_m, \neg P_i \rightarrow$, by lemma 9.2.3, each axiom is labeled with a set of literals

$$\{L_1, \dots, L_m\} \cup \{\neg P_i\} \cup J,$$

where each L_j is in C_j , $1 \leq j \leq m$. In particular, since each clause C_j contains a single positive literal A_j , for every i , $1 \leq i \leq n$, $\{A_1, \dots, A_m\} \cup \{\neg P_i\} \cup J$ must be an axiom. Clearly, either some literal in J is of the form P_i , or there is some definite clause $C = \{\neg Q_1, \dots, \neg Q_p, A_j\}$ among C_1, \dots, C_m , with positive literal $A_j = P_i$. In the first case, $J, C_1, \dots, C_m, \neg P_i \rightarrow$ is an axiom and the tree T_i is not present. Otherwise, let $C' = \{C_1, \dots, C_m\} - \{C\}$. Using lemma 9.2.2 again, we obtain a proof R_i of

$$J, C_1, \dots, C_m, \neg P_i \rightarrow$$

(of depth equal to the previous one) such that the the $\vee : left$ rule is applied to C :

$$\frac{P_i, \neg P_i \rightarrow \quad \frac{T'_i}{J, \{\neg Q_1, \dots, \neg Q_p\}, C', \neg P_i \rightarrow}}{J, \{\neg Q_1, \dots, \neg Q_p, P_i\}, C', \neg P_i \rightarrow}$$

Note that

$$J, \{\neg Q_1, \dots, \neg Q_p\}, C', \neg P_i \rightarrow$$

has two goal clauses. By lemma 9.2.5, either

$$J, \{\neg Q_1, \dots, \neg Q_p\}, C' \rightarrow$$

has a proof U_i , or

$$J, C', \neg P_i \rightarrow$$

has a proof V_i , and the depth of each proof is no greater than the depth of the proof R_i of $J, \{\neg Q_1, \dots, \neg Q_p, P_i\}, C', \neg P_i \rightarrow$. In the second case, by performing a weakening in the last inference of V_i , we obtain a proof for $J, C_1, \dots, C_m, \neg P_i \rightarrow$ of smaller depth than the original, and the induction hypothesis applies, yielding a proof in SLD-form for $J, C_1, \dots, C_m, \neg P_i \rightarrow$. In the first case, $\neg P_i$ is dropped and, by the induction hypothesis, we also have a proof in SLD-form of the form:

$$\frac{P_i, \neg P_i \rightarrow \quad \frac{T'_i}{J, \{\neg Q_1, \dots, \neg Q_p\}, C' \rightarrow}}{J, \{\neg Q_1, \dots, \neg Q_p, P_i\}, C', \neg P_i \rightarrow}$$

Hence, by combining these proofs in SLD-form, we obtain a proof in SLD-form for S . \square

Combining theorem 9.2.1 and lemma 9.2.5, we also have the following theorem.

Theorem 9.2.2 Let S be a set of Horn clauses, consisting of a set J of single positive literals, goal clauses N_1, \dots, N_k , and definite clauses C_1, \dots, C_m containing at least two literals. If S is $GCNF'$ -provable, then there is some i , $1 \leq i \leq k$, such that

$$J \cup \{C_1, \dots, C_m\} \cup \{N_i\}$$

has a $GCNF'$ -proof in SLD-form. \square

Proof: Obvious by theorem 9.2.1 and lemma 9.2.5.

In the next section, we shall show how proofs in SLD-form can be converted into resolution refutations of a certain type.

PROBLEMS

9.2.1. Give a $GCNF'$ -proof in SLD-form for each of the following sequents:

$$\{\neg P_3, \neg P_4, P_5\}, \{\neg P_1, P_2\}, \{\neg P_2, P_1\}, \{\neg P_3, P_4\}, \{P_3\}, \\ \{\neg P_1, \neg P_2\}, \{\neg P_5, P_2\} \rightarrow$$

$$\{P_1\}, \{P_2\}, \{P_3\}, \{P_4\}, \{\neg P_1, \neg P_2, P_6\}, \{\neg P_3, \neg P_4, P_7\}, \\ \{\neg P_6, \neg P_7, P_8\}, \{\neg P_8\} \rightarrow$$

$$\{\neg P_2, P_3\}, \{\neg P_3, P_4\}, \{\neg P_4, P_5\}, \{P_3\}, \{P_1\}, \{P_2\}, \{\neg P_1\}, \\ \{\neg P_3, P_6\}, \{\neg P_3, P_7\}, \{\neg P_3, P_8\} \rightarrow$$

9.2.2. Complete the missing details in the proof of lemma 9.2.5.

9.2.3. Write a computer program for building proof trees in SLD-form for Horn clauses.

* **9.2.4.** Given a set S of Horn clauses, we define an H-tree for S as a tree labeled with propositional letters and satisfying the following properties:

(i) The root of T is labeled with \mathbf{F} (false);

(ii) The immediate descendants of \mathbf{F} are nodes labeled with propositional letters P_1, \dots, P_n such that $\{\neg P_1, \dots, \neg P_n\}$ is some goal clause in S ;

(iii) For every nonroot node in the tree labeled with some letter Q , either the immediate descendants of that node are nodes labeled with letters P_1, \dots, P_k such that $\{\neg P_1, \dots, \neg P_k, Q\}$ is some clause in S , or this node is a leaf if $\{Q\}$ is a clause in S .

Prove that S is unsatisfiable iff it has an H-tree.

9.3 SLD-Resolution in Propositional Logic

SLD-refutations for sets of Horn clauses can be viewed as linearizations of $GCNF'$ -proofs in SLD-form.

9.3.1 SLD-Derivations and SLD-Refutations

First, we show how to linearize SLD-proofs.

Definition 9.3.1 The *linearization procedure* is a recursive algorithm that converts a $GCNF'$ -proof in SLD-form into a sequence of negative clauses according to the following rules:

(1) Every axiom $\neg P, P \rightarrow$ is converted to the sequence $\langle \{\neg P\}, \square \rangle$.

(2) For a sequent $R \rightarrow$ containing a goal clause $N = \{\neg P_1, \dots, \neg P_n\}$, with $n > 1$, if \mathcal{C}_i is the sequence of clauses that is the linearization of the subtree with root the i -th descendant of the sequent $R \rightarrow$, construct the sequence obtained as follows:

Concatenate the sequences $\mathcal{C}'_1, \dots, \mathcal{C}'_{n-1}, \mathcal{C}_n$, where, for each i , $1 \leq i \leq n-1$, letting n_i be the number of clauses in the sequence \mathcal{C}_i , the sequence \mathcal{C}'_i has $n_i - 1$ clauses such that, for every j , $1 \leq j \leq n_i - 1$, if the j -th clause of \mathcal{C}_i is

$$\{B_1, \dots, B_m\},$$

then the j -th clause of \mathcal{C}'_i is

$$\{B_1, \dots, B_m, \neg P_{i+1}, \dots, \neg P_n\}.$$

(3) For every nonaxiom sequent $\Gamma, \neg P \rightarrow$ containing some negative literal $\neg P$, if the definite clause used in the inference is $\{\neg P_1, \dots, \neg P_m, P\}$, letting $\Delta = \Gamma - \{\neg P_1, \dots, \neg P_m, P\}$, then if the sequence of clauses for the sequent $\Delta, \{\neg P_1, \dots, \neg P_m\} \rightarrow$ is \mathcal{C} , form the sequence obtained by concatenating $\neg P$ and the sequence \mathcal{C} .

Note that by (1), (2), and (3), in (2), the first clause of each \mathcal{C}'_i , ($1 \leq i \leq n-1$), is

$$\{\neg P_i, \neg P_{i+1}, \dots, \neg P_n\},$$

and the first clause of \mathcal{C}_n is $\{\neg P_n\}$.

The following example shows how such a linearization is done.

EXAMPLE 9.3.1

Recall the proof tree in SLD-form given in example 9.2.1:

$$\frac{\frac{\frac{P_3, \neg P_3 \rightarrow \quad P_4, \neg P_4 \rightarrow}{\neg P_1, P_1 \rightarrow \quad P_3, P_4, \{\neg P_3, \neg P_4\} \rightarrow} \quad \neg P_2, P_2 \rightarrow \quad P_3, \neg P_3 \rightarrow}{P_3, P_4, \neg P_1, \{\neg P_3, \neg P_4, P_1\} \rightarrow \quad P_3, \neg P_2, \{\neg P_3, P_2\} \rightarrow}}{P_3, P_4, \{\neg P_1, \neg P_2\}, \{\neg P_3, \neg P_4, P_1\}, \{\neg P_3, P_2\} \rightarrow}$$

The sequence corresponding to the left subtree is

$$\langle \{\neg P_1\}, \{\neg P_3, \neg P_4\}, \{\neg P_4\}, \square \rangle$$

and the sequence corresponding to the right subtree is

$$\langle \{\neg P_2\}, \{\neg P_3\}, \square \rangle$$

Hence, the sequence corresponding to the proof tree is

$$\langle \{\neg P_1, \neg P_2\}, \{\neg P_3, \neg P_4, \neg P_2\}, \{\neg P_4, \neg P_2\}, \{\neg P_2\}, \{\neg P_3\}, \square \rangle.$$

This last sequence is an SLD-refutation, as defined below.

Definition 9.3.2 Let S be a set of Horn clauses consisting of a set D of definite clauses and a set $\{G_1, \dots, G_q\}$ of goals. An *SLD-derivation* for S is a sequence $\langle N_0, N_1, \dots, N_p \rangle$ of negative clauses satisfying the following properties:

- (1) $N_0 = G_j$, where G_j is one of the goals;
- (2) For every N_i in the sequence, $0 \leq i < p$, if

$$N_i = \{\neg A_1, \dots, \neg A_{k-1}, \neg A_k, \neg A_{k+1}, \dots, \neg A_n\},$$

then there is some definite clause

$$C_i = \{\neg B_1, \dots, \neg B_m, A_k\}$$

in D such that, if $m > 0$, then

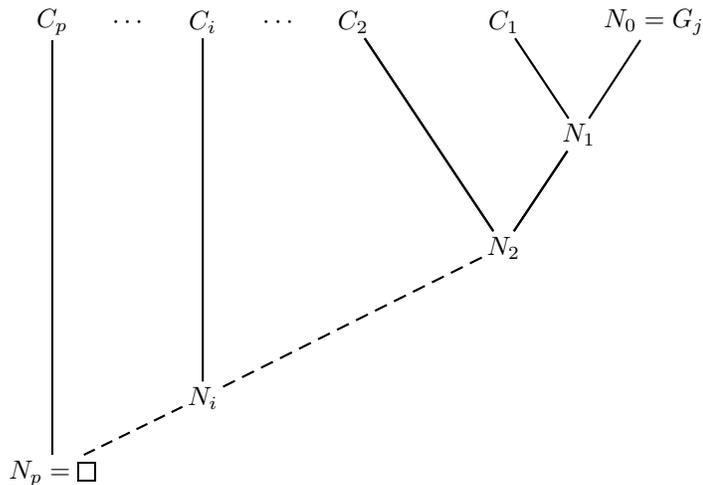
$$N_{i+1} = \{\neg A_1, \dots, \neg A_{k-1}, \neg B_1, \dots, \neg B_m, \neg A_{k+1}, \dots, \neg A_n\}$$

else if $m = 0$ then

$$N_{i+1} = \{\neg A_1, \dots, \neg A_{k-1}, \neg A_{k+1}, \dots, \neg A_n\}.$$

An SLD-derivation is an *SLD-refutation* iff $N_p = \square$. The *SLD-resolution method* is the method in which a set of Horn clauses is shown to be unsatisfiable by finding an SLD-refutation.

Note that an SLD-derivation is a linear representation of a resolution DAG of the following special form:



At each step, the clauses

$$\{\neg A_1, \dots, \neg A_{k-1}, \neg A_k, \neg A_{k+1}, \dots, \neg A_n\} \quad \text{and} \\ \{\neg B_1, \dots, \neg B_m, A_k\}$$

are resolved, the literals A_k and $\neg A_k$ being canceled. The literal A_k is called the *selected atom* of N_i , and the clauses N_0, C_1, \dots, C_p are the *input clauses*.

Such a resolution method is a form of linear input resolution, because it resolves the current clause N_k with some clause in the input set D .

By the soundness of the resolution method (lemma 4.3.2), the SLD-resolution method is sound.

EXAMPLE 9.3.2

The sequence

$$\begin{aligned} &< \{\neg P_1, \neg P_2\}, \{\neg P_3, \neg P_4, \neg P_2\}, \\ &\{\neg P_4, \neg P_2\}, \{\neg P_2\}, \{\neg P_3\}, \square > \end{aligned}$$

of example 9.3.1 is an SLD-refutation.

9.3.2 Completeness of SLD-Resolution for Horn Clauses

In order to show that SLD-resolution is complete for Horn clauses, since by theorem 9.2.2 every set of Horn clauses has a $GCNF'$ -proof in SLD-form, it is sufficient to prove that the linearization algorithm of definition 9.3.1 converts a proof in SLD-form to an SLD-refutation.

Lemma 9.3.1 (Correctness of the linearization process) Given any $GCNF'$ -proof T in SLD-form, the linearization procedure outputs an SLD-refutation.

Proof: We proceed by induction on proofs. If T consists of an axiom, then the set S of Horn clauses contains a goal $\neg Q$ and a positive literal Q , and we have the SLD-refutation $\langle \{\neg Q\}, \square \rangle$.

Otherwise, because it is in SLD-form, letting $\mathcal{C} = C_1, \dots, C_m$, the tree T has the following structure:

$$\begin{array}{c} \frac{\frac{T_{n-1}}{J, \mathcal{C}, \neg P_{n-1} \rightarrow} \quad \frac{T_n}{J, \mathcal{C}, \neg P_n \rightarrow}}{J, \mathcal{C}, \{\neg P_{n-1}, \neg P_n\} \rightarrow} \\ \dots \\ \frac{\frac{T_1}{J, \mathcal{C}, \neg P_1 \rightarrow} \quad \frac{\frac{T_2}{J, \mathcal{C}, \neg P_2 \rightarrow} \quad J, \mathcal{C}, \{\neg P_3, \dots, \neg P_n\} \rightarrow}{J, \mathcal{C}, \{\neg P_2, \dots, \neg P_n\} \rightarrow}}{J, \mathcal{C}, \{\neg P_1, \dots, \neg P_n\} \rightarrow} \end{array}$$

Each tree T_i that is not an axiom is also in SLD-form and has the following shape:

$$\frac{P_i, \neg P_i \rightarrow \quad \frac{T'_i}{J, \{\neg Q_1, \dots, \neg Q_p\}, C' \rightarrow}}{J, \{\neg Q_1, \dots, \neg Q_p, P_i\}, C', \neg P_i \rightarrow}$$

where $C' = \{C_1, \dots, C_m\} - \{C\}$, for some definite clause $C = \{\neg Q_1, \dots, \neg Q_p, P_i\}$.

By the induction hypothesis, each tree T'_i is converted to an SLD-refutation

$$Y_i = \langle \{\neg Q_1, \dots, \neg Q_p\}, N_2, \dots, N_q \rangle .$$

By rule (3), the proof tree T_i is converted to the SLD-refutation X_i obtained by concatenating $\{\neg P_i\}$ and Y_i . But then,

$$X_i = \langle \{\neg P_i\}, \{\neg Q_1, \dots, \neg Q_p\}, N_2, \dots, N_q \rangle$$

is an SLD-refutation obtained by resolving $\{\neg P_i\}$ with $\{\neg Q_1, \dots, \neg Q_p, P_i\}$.

If T_i is an axiom then by rule (1) it is converted to $\langle \{\neg P_i\}, \square \rangle$, which is an SLD-refutation.

Finally, rule (2) combines the SLD-refutations X_1, \dots, X_n in such a way that the resulting sequence is an SLD-refutation. Indeed, for every i , $1 \leq i \leq n-1$, X_i becomes the SLD-derivation X'_i , where

$$X'_i = \langle \{\neg P_i, \neg P_{i+1}, \dots, \neg P_n\}, \{\neg Q_1, \dots, \neg Q_p, \neg P_{i+1}, \dots, \neg P_n\}, N_2 \cup \{\neg P_{i+1}, \dots, \neg P_n\}, \dots, N_{q-1} \cup \{\neg P_{i+1}, \dots, \neg P_n\} \rangle ,$$

and so the entire sequence $X'_1, \dots, X'_{n-1}, X_n$ is an SLD-refutation starting from the goal $\{\neg P_1, \dots, \neg P_n\}$. \square

As a corollary, we have the completeness of SLD-resolution for Horn clauses.

Theorem 9.3.1 (Completeness of SLD-resolution for Horn clauses) The SLD-resolution method is complete for Horn clauses. Furthermore, if the first negative clause is $\{\neg P_1, \dots, \neg P_n\}$, for every literal $\neg P_i$ in this goal, there is an SLD-resolution whose first selected atom is P_i .

Proof: Completeness is a consequence of lemma 9.3.1 and theorem 9.2.2. It is easy to see that in the linearization procedure, the order in which the subsequences are concatenated does not matter. This implies the second part of the lemma. \square

Actually, since SLD-refutations are the result of linearizing proof trees in SLD-form, it is easy to show that any atom P_i such that $\neg P_i$ belongs to a negative clause N_k in an SLD-refutation can be chosen as the selected atom.

By theorem 9.2.2, if a set S of Horn clauses with several goals N_1, \dots, N_k is $GCNF'$ -provable, then there is some goal N_i such that $S - \{N_1, \dots, N_{i-1}, N_{i+1}, \dots, N_k\}$ is $GCNF'$ -provable. This does not mean that there is a unique such N_i , as shown by the following example.

EXAMPLE 9.3.2

Consider the set S of clauses:

$$\{P\}, \{Q\}, \{\neg S, R\}, \{\neg R, \neg P\}, \{\neg R, \neg Q\}, \{S\}.$$

We have two SLD-refutations:

$$\langle \{\neg R, \neg P\}, \{\neg R\}, \{\neg S\}, \square \rangle$$

and

$$\langle \{\neg R, \neg Q\}, \{\neg R\}, \{\neg S\}, \square \rangle.$$

In the next section, we generalize SLD-resolution to first-order languages without equality, using the lifting technique of Section 8.5.

PROBLEMS

9.3.1. Apply the linearization procedure to the proof trees in SLD-form obtained in problem 9.2.1.

9.3.2. Give different SLD-resolution refutations for the following sets of clauses:

$$\{P_1\}, \{P_2\}, \{P_3\}, \{P_4\}, \{\neg P_1, \neg P_2, P_6\}, \{\neg P_3, \neg P_4, P_7\}, \\ \{\neg P_6, \neg P_7, P_8\}, \{\neg P_8\}.$$

$$\{\neg P_2, P_3\}, \{\neg P_3, P_4\}, \{\neg P_4, P_5\}, \{P_3\}, \{P_1\}, \{P_2\}, \{\neg P_1\}, \\ \{\neg P_3, P_6\}, \{\neg P_3, P_7\}, \{\neg P_3, P_8\}.$$

9.3.3. Write a computer program implementing the linearization procedure.

9.4 SLD-Resolution in First-Order Logic

In this section we shall generalize SLD-resolution to first-order languages without equality. Fortunately, it is relatively painless to generalize results about

propositional SLD-resolution to the first-order case, using the lifting technique of Section 8.5.

9.4.1 Definition of SLD-Refutations

Since the main application of SLD-resolution is to PROLOG, we shall also revise our notation to conform to the PROLOG notation.

Definition 9.4.1 A *Horn clause* (in PROLOG notation) is one of the following expressions:

(i) $B : -A_1, \dots, A_m$

(ii) B

(iii) $: -A_1, \dots, A_m$

In the above, B, A_1, \dots, A_m are atomic formulae of the form $Pt_1 \dots t_k$, where P is a predicate symbol of rank k , and t_1, \dots, t_k are terms.

A clause of the form (i) or (ii) is called a *definite clause*, and a clause of the form (iii) is called a *goal clause* (or *negative clause*).

The translation into the standard logic notation is the following:

The clause $B : -A_1, \dots, A_m$ corresponds to the formula

$$(\neg A_1 \vee \dots \vee \neg A_m \vee B);$$

The clause B corresponds to the atomic formula B ;

The clause $: -A_1, \dots, A_m$ corresponds to the formula

$$(\neg A_1 \vee \dots \vee \neg A_m).$$

Actually, as in definition 8.2.1, it is assumed that a Horn clause is the universal closure of a formula as above (that is, of the form $\forall x_1 \dots \forall x_n C$, where $FV(C) = \{x_1, \dots, x_n\}$). The universal quantifiers are dropped for simplicity of notation, but it is important to remember that they are implicitly present.

The definition of SLD-derivations and SLD-refutations is extended by combining definition 9.3.2 and the definition of a resolvent given in definition 8.5.2.

Definition 9.4.2 Let S be a set of Horn clauses consisting of a set D of definite clauses and a set $\{G_1, \dots, G_q\}$ of goals. An *SLD-derivation* for S is a sequence $\langle N_0, N_1, \dots, N_p \rangle$ of negative clauses satisfying the following properties:

- (1) $N_0 = G_j$, where G_j is one of the goals;

(2) For every N_i in the sequence, $0 \leq i < p$, if

$$N_i =: -A_1, \dots, A_{k-1}, A_k, A_{k+1}, \dots, A_n,$$

then there is some definite clause $C_i = A : -B_1, \dots, B_m$ in D such that A_k and A are unifiable, and for some most general unifier σ_i of A_k and $\rho_i(A)$, where (Id, ρ_i) is a separating substitution pair, if $m > 0$, then

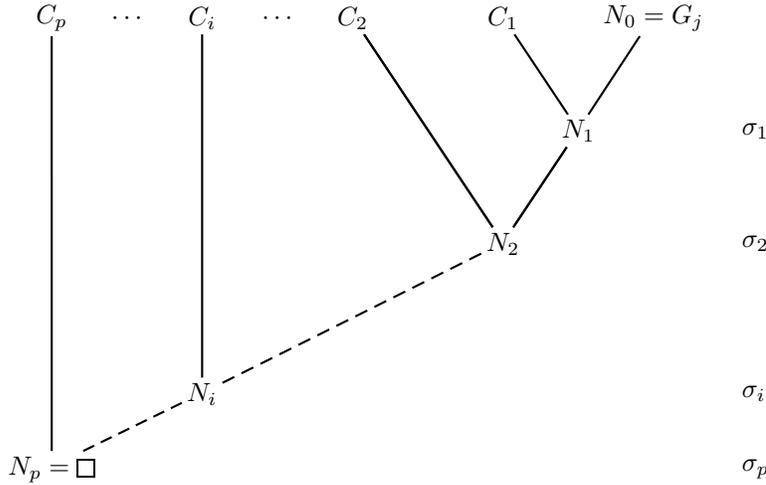
$$N_{i+1} =: -\sigma_i(A_1, \dots, A_{k-1}, \rho_i(B_1), \dots, \rho_i(B_m), A_{k+1}, \dots, A_n)$$

else if $m = 0$ then

$$N_{i+1} =: -\sigma_i(A_1, \dots, A_{k-1}, A_{k+1}, \dots, A_n).$$

An SLD-derivation is an *SLD-refutation* iff $N_p = \square$.

Note that an SLD-derivation is a linear representation of a resolution DAG of the following special form:



At each step, the clauses

$$: -A_1, \dots, A_{k-1}, A_k, A_{k+1}, \dots, A_n$$

and

$$A : -B_1, \dots, B_m$$

are resolved, the atoms A_k and $\rho_i(A)$ being canceled, since they are unified by the most general unifier σ_i . The literal A_k is called the *selected atom* of N_i , and the clauses N_0, C_1, \dots, C_p are the *input clauses*.

When the derivation is a refutation, the substitution

$$\sigma = (\rho_1 \circ \sigma_1) \circ \dots \circ (\rho_p \circ \sigma_p)$$

obtained by composing the substitutions occurring in the refutation is called the *result substitution* or *answer substitution*. It is used in PROLOG to extract the output of an SLD-computation.

Since an SLD-derivation is a special kind of resolution DAG, (a linear input resolution), its soundness is a consequence of lemma 8.5.2.

Lemma 9.4.1 (Soundness of SLD-resolution) If a set of Horn clauses has an SLD-refutation, then it is unsatisfiable.

Proof: Immediate from lemma 8.5.2. \square

Let us give an example of an SLD-refutation in the first-order case.

EXAMPLE 9.4.1

Consider the following set of definite clauses, axiomatizing addition of natural numbers:

$$\begin{aligned} C_1 &: \text{add}(X, 0, X). \\ C_2 &: \text{add}(X, \text{succ}(Y), \text{succ}(Z)) : \neg \text{add}(X, Y, Z). \end{aligned}$$

Consider the goal

$$B : \neg \text{add}(\text{succ}(0), V, \text{succ}(\text{succ}(0))).$$

We wish to show that the above set is unsatisfiable. We have the following SLD-refutation:

| Goal clause | Input clause | Substitution |
|---|--------------|--------------|
| $: \neg \text{add}(\text{succ}(0), V, \text{succ}(\text{succ}(0)))$ | C_2 | |
| $: \neg \text{add}(\text{succ}(0), Y_2, \text{succ}(0))$ | C_1 | σ_1 |
| \square | | σ_2 |

where

$$\begin{aligned} \sigma_1 &= (\text{succ}(0)/X_1, \text{succ}(0)/Z_1, \text{succ}(Y_2)/V), \\ \sigma_2 &= (\text{succ}(0)/X_2, 0/Y_2) \end{aligned}$$

The variables X_1, Z_1, Y_2, X_2 were introduced by separating substitutions in computing resolvents. The result substitution is

$$(\text{succ}(0)/V, \text{succ}(0)/X_1, \text{succ}(0)/Z_1, \text{succ}(0)/X_2).$$

The interesting component is $\text{succ}(0)/V$. Indeed, there is a computational interpretation of the unsatisfiability of the set $\{C_1, C_2, B\}$. For this, it is necessary to write quantifiers explicitly and remember that goal clauses are negative. Observe that

$$\forall X C_1 \wedge \forall X Y \forall Z C_2 \wedge \forall V B$$

is unsatisfiable, iff

$$\neg(\forall X C_1 \wedge \forall X \forall Y \forall Z C_2 \wedge \forall V B)$$

is valid, iff

$$(\forall X C_1 \wedge \forall X \forall Y \forall Z C_2) \supset \exists V \neg B$$

is valid. But $\exists V \neg B$ is actually

$$\exists V \text{add}(\text{succ}(0), V, \text{succ}(0)).$$

Since $(\forall X C_1 \wedge \forall X \forall Y \forall Z C_2)$ defines addition in the intuitive sense that any X, Y, Z satisfying the above sentence are such that $Z = X + Y$, we are trying to find some V such that $\text{succ}(0) + V = \text{succ}(\text{succ}(0))$, or in other words, compute the difference of $\text{succ}(\text{succ}(0))$ and $\text{succ}(0)$, which is indeed $\text{succ}(0)$!

This interpretation of a refutation showing that a set of Horn clauses is unsatisfiable as a computation of the answer to a query, such as

$$(\forall X C_1 \wedge \forall X \forall Y \forall Z C_2) \supset \exists V \neg B,$$

“find some V satisfying $\neg B$ and such that some conditional axioms $\forall X C_1$ and $\forall X \forall Y \forall Z C_2$ hold,”

is the essence of PROLOG. The set of clauses $\{C_1, C_2\}$ can be viewed as a *logic program*.

We will come back to the idea of refutations as computations in the next section.

9.4.2 Completeness of SLD-Resolution for Horn Clauses

The completeness of SLD-resolution for Horn clauses is shown in the following theorem.

Theorem 9.4.1 (Completeness of SLD-Resolution for Horn Clauses) Let \mathbf{L} be any first-order language without equality. Given any finite set S of Horn clauses, if S is unsatisfiable, then there is an SLD-refutation with first clause some negative clause $: -B_1, \dots, B_n$ in S .

Proof: We shall use the lifting technique provided by lemma 8.5.4. First, by the Skolem-Herbrand-Gödel theorem, if S is unsatisfiable, there is a set S_g of ground instances of clauses in S which is unsatisfiable. Since substitution instances of Horn clauses are Horn clauses, by theorem 9.3.1, there is an SLD-refutation for S_g , starting from some negative clause in S_g . Finally, we conclude by observing that if we apply the lifting technique of lemma 8.5.4, we obtain an SLD-refutation. This is because we always resolve a negative clause (N_i) against an input clause (C_i). Hence, the result is proved. \square

From theorem 9.3.1, it is also true that if the first negative clause is $: -B_1, \dots, B_n$, for every atom B_i in this goal, there is an SLD-resolution whose first selected atom is B_i . As a matter of fact, this property holds for any clause N_i in the refutation.

Even though SLD-resolution is complete for Horn clauses, there is still the problem of choosing among many possible SLD-derivations. The above shows that the choice of the selected atom is irrelevant. However, we still have the problem of choosing a definite clause $A : -B_1, \dots, B_m$ such that A unifies with one of the atoms in the current goal clause $: -A_1, \dots, A_{k-1}, A_k, A_{k+1}, \dots, A_n$.

Such problems are important and are the object of current research in programming logic, but we do not have the space to address them here. The interested reader is referred to Kowalski, 1979, or Campbell, 1983, for an introduction to the methods and problems in programming logic.

In the next section, we discuss the use of SLD-resolution as a computation procedure for PROLOG.

PROBLEMS

- 9.4.1.** Prove using SLD-resolution that the following set of clauses is unsatisfiable:

$$\begin{aligned} &add(X, 0, X) \\ &add(X, succ(Y), succ(Z)) : -add(X, Y, Z) \\ &: -add(succ(succ(0)), succ(succ(0)), U). \end{aligned}$$

- 9.4.2.** Prove using SLD-resolution that the following set of clauses is unsatisfiable:

$$\begin{aligned} &add(X, 0, X) \\ &add(X, succ(Y), succ(Z)) : -add(X, Y, Z) \\ &: -add(U, V, succ(succ(succ(0)))). \end{aligned}$$

Find all possible SLD-refutations.

- 9.4.3.** Using SLD-resolution, show that the following set of Horn clauses is unsatisfiable:

$$\begin{aligned} &hanoi(N, Output) : -move(a, b, c, N, Output). \\ &move(A, B, C, succ(M), Output) : -move(A, C, B, M, Out1), \\ &move(C, B, A, M, Out2), \\ &append(Out1, cons(to(A, B), Out2), Output). \\ &move(A, B, C, 0, nil). \\ &append(cons(A, L1), L2, cons(A, L3)) : -append(L1, L2, L3). \\ &append(nil, L1, L1). \\ &: -hanoi(succ(succ(0)), Z) \end{aligned}$$

9.5 SLD-Resolution, Logic Programming (PROLOG)

We have seen in example 9.4.1 that an SLD-refutation for a set of Horn clauses can be viewed as a computation. This illustrates an extremely interesting use of logic as a *programming language*.

9.5.1 Refutations as Computations

In the past few years, Horn logic has been the basis of a new type of programming language due to Colmerauer named PROLOG. It is not the purpose of this book to give a complete treatment of PROLOG, and we refer the interested reader to Kowalski, 1979, or Clocksin and Mellish, 1981, for details. In this section, we shall lay the foundations of the programming logic PROLOG. It will be shown how SLD-resolution can be used as a computational procedure to solve certain problems, and the correctness and completeness of this approach will be proved.

In a logic programming language like PROLOG, one writes programs as sets of assertions in the form of Horn clauses, or more accurately, definite clauses, except for the goal. A set P of definite clauses is a *logic program*. As we said in Section 9.4, it is assumed that distinct Horn clauses are universally quantified.

Roughly speaking, a logic program consists of facts and assertions. Given such a logic program, one is usually interested in extracting facts that are consequences of the logic program P . Typically, one has a certain “query” (or goal) G containing some free variables z_1, \dots, z_q , and one wants to find term instances t_1, \dots, t_q for the variables z_1, \dots, z_q , such that the formula

$$P \supset G[t_1/z_1, \dots, t_q/z_q]$$

is valid.

For simplicity, it will be assumed that the query is a positive atomic formula G . More complicated formulae can be handled (anti-Horn clauses), but we will consider this case later. In PROLOG, a goal statement G is denoted by $? - G$.

From a logical point of view, the problem is to determine whether the sentence

$$P \supset (\exists z_1 \dots \exists z_q G)$$

is valid.

From a computational point of view, the problem is to find term values t_1, \dots, t_q for the variables z_1, \dots, z_q that make the formula

$$P \supset G[t_1/z_1, \dots, t_q/z_q]$$

valid, and perhaps all such assignments.

Remarkably, SLD-resolution can be used not only as a proof procedure, but also a computational procedure, because it returns a result substitution. The reason is as follows:

The formula $P \supset (\exists z_1 \dots \exists z_q G)$ is valid iff
 $\neg(P \supset (\exists z_1 \dots \exists z_q G))$ is unsatisfiable iff
 $P \wedge (\forall z_1 \dots \forall z_q \neg G)$ is unsatisfiable.

But since G is an atomic formula, $\neg G$ is a goal clause : $\neg G$, and $P \wedge (\forall z_1 \dots \forall z_q \neg G)$ is a conjunction of Horn clauses!

Hence, SLD-resolution can be used to test for unsatisfiability, and if it succeeds, it returns a result substitution σ . The crucial fact is that the components of the substitution σ corresponding to the variables z_1, \dots, z_q are answers to the query G . However, this fact is not obvious. A proof will be given in the next section. As a preliminary task, we give a rigorous definition of the semantics of a logic program.

9.5.2 Model-Theoretic Semantics of Logic Programs

We begin by defining what kind of formula can appear as a goal.

Definition 9.5.1 An *anti-Horn clause* is a formula of the form

$$\exists x_1 \dots \exists x_m B,$$

where B is a conjunction of literals $L_1 \wedge \dots \wedge L_p$, with at most one negative literal and $FV(B) = \{x_1, \dots, x_m\}$.

A *logic program* is a pair (P, G) , where the *program* P is a set of (universal) Horn clauses, and the *query* G is a disjunction

$$(G_1 \vee \dots \vee G_n)$$

of anti-Horn clauses $G_i = \exists y_1 \dots \exists z_{m_i} B_i$.

It is also assumed that for all $i \neq j$, $1 \leq i, j \leq n$, the sets of variables $FV(B_i)$ and $FV(B_j)$ are disjoint. The union $\{z_1, \dots, z_q\}$ of the sets of free variables occurring in each B_i is called the set of *output variables associated with G* .

Note that an anti-Horn clause is not a clause. However, the terminology is justified by the fact that the negation of an anti-Horn clause is a (universal) Horn clause, and that $\neg G$ is equivalent to a conjunction of universal Horn clauses.

Remark: This definition is more general than the usual definition used in PROLOG. In (standard) PROLOG, P is a set of definite clauses (that is,

P does not contain negative clauses), and G is a formula that is a conjunction of atomic formulae. It is shown in the sequel that more general queries can be handled, but that the semantics is a bit more subtle. Indeed, indefinite answers may arise.

EXAMPLE 9.5.1

The following is a logic program, where P consists of the following clauses:

$$\begin{aligned} & \text{rocksinger}(\text{jackson}). \\ & \text{teacher}(\text{jean}). \\ & \text{teacher}(\text{susan}). \\ & \text{rich}(X) : \neg \text{rocksinger}(X). \\ & \quad : \neg \text{teacher}(X), \text{rich}(X). \end{aligned}$$

The query is the following disjunction:

$$? - \neg \text{rocksinger}(Y) \vee \text{rich}(Z)$$
EXAMPLE 9.5.2

The following is the program of a logic program:

$$\begin{aligned} & \text{hanoi}(N, \text{Output}) : \neg \text{move}(a, b, c, N, \text{Output}). \\ & \text{move}(A, B, C, \text{succ}(M), \text{Output}) : \neg \text{move}(A, C, B, M, \text{Out1}), \\ & \quad \text{move}(C, B, A, M, \text{Out2}), \\ & \quad \text{append}(\text{Out1}, \text{cons}(to(A, B), \text{Out2}), \text{Output}). \\ & \text{move}(A, B, C, 0, \text{nil}). \\ & \text{append}(\text{cons}(A, L1), L2, \text{cons}(A, L3)) : \neg \text{append}(L1, L2, L3). \\ & \text{append}(\text{nil}, L1, L1). \end{aligned}$$

The query is:

$$? - \text{hanoi}(\text{succ}(\text{succ}(\text{succ}(0))), \text{Output}).$$

The above program is a logical version of the well known problem known as the tower of Hanoi (see Clocksin and Mellish, 1981).

In order to give a rigorous definition of the semantics of a logic program, it is convenient to define the concept of a free structure. Recall that we are only dealing with first-order languages without equality, and that if the language has no constants, the special constant $\#$ is added to it.

Definition 9.5.2 Given a first-order language \mathbf{L} without equality and with at least one constant, a *free structure* (or *Herbrand structure*) \mathbf{H} is an \mathbf{L} -structure with domain the set $H_{\mathbf{L}}$ of all closed \mathbf{L} -terms, and whose interpretation function satisfies the following property:

(i) For every function symbol f of rank n , for all $t_1, \dots, t_n \in H_{\mathbf{L}}$,

$$f_{\mathbf{H}}(t_1, \dots, t_n) = ft_1 \dots t_n \quad \text{and}$$

(ii) For every constant symbol c ,

$$c_{\mathbf{H}} = c.$$

The set of terms $H_{\mathbf{L}}$ is called the *Herbrand universe* of \mathbf{L} . For simplicity of notation, the set $H_{\mathbf{L}}$ is denoted as H when \mathbf{L} is understood. The following lemma shows that free structures are universal. This lemma is actually not necessary for giving the semantics of Horn clauses, but it is of independent interest.

Lemma 9.5.1 A sentence X in NNF containing only universal quantifiers is satisfiable in some model iff it is satisfiable in some free structure.

Proof: Clearly, if X is satisfied in a free structure, it is satisfiable in some model. For the converse, assume that X has some model \mathbf{A} . We show how a free structure can be constructed from \mathbf{A} . We define the function $h : H \rightarrow A$ as follows:

For every constant c , $h(c) = c_{\mathbf{A}}$;

For every function symbol f of rank $n > 0$, for any n terms $t_1, \dots, t_n \in H$,

$$h(ft_1 \dots t_n) = f_{\mathbf{A}}(h(t_1), \dots, h(t_n)).$$

Define the interpretation of the free structure H such that, for any predicate symbol P of rank n , for any n terms $t_1, \dots, t_n \in H$,

$$\mathbf{H} \models P(t_1, \dots, t_n) \quad \text{iff} \quad \mathbf{A} \models P(h(t_1), \dots, h(t_n)). \quad (*)$$

We now prove by induction on formulae that, for every assignment $s : \mathbf{V} \rightarrow H$, if $\mathbf{A} \models X[s \circ h]$, then $\mathbf{H} \models X[s]$.

(i) If X is a literal, this amounts to the definition (*).

(ii) If X is of the form $(B \wedge C)$, then $\mathbf{A} \models X[s \circ h]$ implies that

$$\mathbf{A} \models B[s \circ h] \quad \text{and} \quad \mathbf{A} \models C[s \circ h].$$

By the induction hypothesis,

$$\mathbf{H} \models B[s] \quad \text{and} \quad \mathbf{H} \models C[s],$$

that is, $\mathbf{H} \models X[s]$.

(iii) If X is of the form $(B \vee C)$, then $\mathbf{A} \models X[s \circ h]$ implies that

$$\mathbf{A} \models B[s \circ h] \quad \text{or} \quad \mathbf{A} \models C[s \circ h].$$

By the induction hypothesis,

$$\mathbf{H} \models B[s] \quad \text{or} \quad \mathbf{H} \models C[s],$$

that is, $\mathbf{H} \models X[s]$.

(iv) X is of the form $\exists xB$. This case is not possible since X does not contain existential quantifiers.

(v) X is of the form $\forall xB$. If $\mathbf{A} \models X[s \circ h]$, then for every $a \in A$,

$$\mathbf{A} \models B[(s \circ h)[x := a]].$$

Now, since $h : H \rightarrow A$, for every $t \in H$, $h(t) = a$ for some $a \in A$, and so, for every t in H ,

$$\mathbf{A} \models B[(s \circ h)[x := h(t)]], \text{ that is, } \mathbf{A} \models B[(s[x := t]) \circ h].$$

By the induction hypothesis, $\mathbf{H} \models B[s[x := t]]$ for all $t \in H$, that is, $\mathbf{H} \models X[s]$.
□

It is obvious that lemma 9.5.1 also applies to sets of sentences. Also, since a formula is unsatisfiable iff it has no model, we have the following corollary:

Corollary Given a first-order language without equality and with some constant, a set of sentences in NNF and only containing universal quantifiers is unsatisfiable iff it is unsatisfiable in every free (Herbrand) structure. □

We now provide a rigorous semantics of logic programs.

Given a logic program (P, G) , the question of interest is to determine whether the formula $P \supset G$ is valid. Actually, we really want more. If $\{z_1, \dots, z_q\}$ is the set of output variables occurring in G , we would like to find some (or all) tuple(s) (t_1, \dots, t_q) of ground terms such that

$$\models P \supset (B_1 \vee \dots \vee B_n)[t_1/z_1, \dots, t_q/z_q].$$

As we shall see, such tuples do not always exist. However, indefinite (or disjunctive) answers always exist, and if some conditions are imposed on P and G , definite answers (tuples of ground terms) exist.

Assume that $P \supset G$ is valid. This is equivalent to $\neg(P \supset G)$ being unsatisfiable. But $\neg(P \supset G)$ is equivalent to $P \wedge \neg G$, which is equivalent to a conjunction of universal Horn clauses. By the Skolem-Herbrand-Gödel

theorem (theorem 7.6.1), if $\{x_1, \dots, x_m\}$ is the set of all universally quantified variables in $P \wedge \neg G$, there is *some set*

$$\{(t_1^1, \dots, t_m^1), \dots, (t_1^k, \dots, t_m^k)\}$$

of m -tuples of ground terms such that the conjunction

$$(P \wedge \neg G)[t_1^1/x_1, \dots, t_m^1/x_m] \wedge \dots \wedge (P \wedge \neg G)[t_1^k/x_1, \dots, t_m^k/x_m]$$

is unsatisfiable (for some $k \geq 1$). From this, it is not difficult to prove that

$$\models P \supset G[t_1^1/x_1, \dots, t_m^1/x_m] \vee \dots \vee G[t_1^k/x_1, \dots, t_m^k/x_m].$$

However, we cannot claim that $k = 1$, as shown by the following example.

EXAMPLE 9.5.3

Let $P = \neg Q(a) \vee \neg Q(b)$, and $G = \exists x \neg Q(x)$. $P \supset G$ is valid, but there is no term t such that

$$\neg Q(a) \vee \neg Q(b) \supset \neg Q(t)$$

is valid.

As a consequence, the answer to a query may be indefinite, in the sense that it is a disjunction of substitution instances of the goal. However, definite answers can be ensured if certain restrictions are met.

Lemma 9.5.2 (Definite answer lemma) If P is a (finite) set of definite clauses and G is a query of the form

$$\exists z_1 \dots \exists z_q (B_1 \wedge \dots \wedge B_l),$$

where each B_i is an atomic formula, if

$$\models P \supset \exists z_1 \dots \exists z_q (B_1 \wedge \dots \wedge B_l),$$

then there is some tuple (t_1, \dots, t_q) of ground terms such that

$$\models P \supset (B_1 \wedge \dots \wedge B_l)[t_1/z_1, \dots, t_q/z_q].$$

Proof:

$$\begin{aligned} & \models P \supset \exists z_1 \dots \exists z_q (B_1 \wedge \dots \wedge B_l) \quad \text{iff} \\ & P \wedge \forall z_1 \dots \forall z_q (\neg B_1 \vee \dots \vee \neg B_l) \quad \text{is unsatisfiable.} \end{aligned}$$

By the Skolem-Herbrand-Gödel theorem, there is a set C of ground substitution instances of the clauses in $P \cup \{\neg B_1, \dots, \neg B_l\}$ that is unsatisfiable. Since

the only negative clauses in C come from $\{\neg B_1, \dots, \neg B_l\}$, by lemma 9.2.5, there is some substitution instance

$$(\neg B_1 \vee \dots \vee \neg B_l)[t_1/z_1, \dots, t_q/z_q]$$

such that

$$P' \cup \{(\neg B_1 \vee \dots \vee \neg B_l)[t_1/z_1, \dots, t_q/z_q]\}$$

is unsatisfiable, where P' is the subset of C consisting of substitution instances of clauses in P . But then, it is not difficult to show that

$$\models P \supset (B_1 \wedge \dots \wedge B_l)[t_1/z_1, \dots, t_q/z_q]. \quad \square$$

The result of lemma 9.5.2 justifies the reason that in PROLOG only programs consisting of definite clauses and queries consisting of conjunctions of atomic formulae are considered. With such restrictions, definite answers are guaranteed. The above discussion leads to the following definition.

Definition 9.5.3 Given a logic program (P, G) with query $G = \exists z_1 \dots \exists z_q B$ and with $B = (B_1 \vee \dots \vee B_n)$, the *semantics* (or *meaning*) of (P, G) is the set

$$M(P, G) = \bigcup \{ \{ (t_1^1, \dots, t_q^1), \dots, (t_1^k, \dots, t_q^k) \}, k \geq 1, (t_1^k, \dots, t_q^k) \in H^q \mid \models P \supset B[t_1^1/z_1, \dots, t_q^1/z_q] \vee \dots \vee B[t_1^k/z_1, \dots, t_q^k/z_q] \}$$

of sets q -tuples of terms in the Herbrand universe H that make the formula

$$P \supset B[t_1^1/z_1, \dots, t_q^1/z_q] \vee \dots \vee B[t_1^k/z_1, \dots, t_q^k/z_q]$$

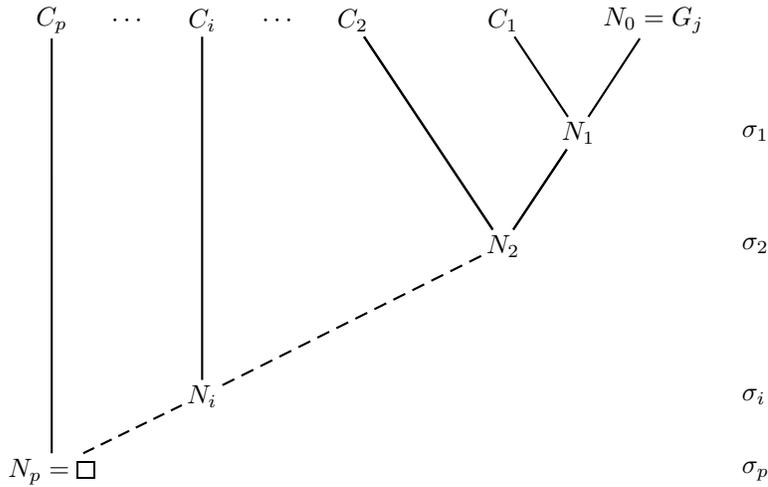
valid (in every free structure).

If P is a set of definite clauses and B is a conjunction of atomic formulae, $k = 1$.

9.5.3 Correctness of SLD-Resolution as a Computation Procedure

We now prove that for every SLD-refutation of the conjunction of clauses in $P \wedge \neg G$, the components of the result substitution σ restricted to the output variables belong to the semantics $M(P, G)$ of (P, G) . We prove the following slightly more general lemma, which implies the fact mentioned above.

Lemma 9.5.3 Given a set P of Horn clauses, let \mathcal{R} be an SLD-refutation



with result substitution σ (not necessarily ground). Let $\theta_p = \rho_p \circ \sigma_p$, and for every i , $1 \leq i \leq p-1$, let

$$\theta_i = (\rho_i \circ \sigma_i) \circ \theta_{i+1}.$$

(Note that $\sigma = \theta_1$, the result substitution.) The substitutions θ_i are also called *result substitutions*.) Then the set of quantifier-free clauses

$$\{\theta_1(N_0), \theta_1(C_1), \dots, \theta_p(C_p)\}$$

is unsatisfiable (using the slight abuse of notation in which the matrix D of a clause $C = \forall x_1 \dots \forall x_k D$ is also denoted by C).

Proof: We proceed by induction on the length of the derivation.

(i) If $p = 1$, N_0 must be a negative formula: $-B$ and C_1 a positive literal A such that A and B are unifiable, and it is clear that $\{-\theta_1(B), \theta_1(C_1)\}$ is unsatisfiable.

(ii) If $p > 1$, then by the induction hypothesis, taking N_1 as the goal of an SLD-refutation of length $p-1$ the set

$$\{\theta_2(N_1), \theta_2(C_2), \dots, \theta_p(C_p)\}$$

is unsatisfiable. But N_0 is some goal clause

$$: -A_1, \dots, A_{k-1}, A_k, A_{k+1}, \dots, A_n,$$

and C_1 is some definite clause

$$A : -B_1, \dots, B_m,$$

such that A and A_k are unifiable. Furthermore, the resolvent is given by

$$N_1 =: -\sigma_1(A_1, \dots, A_{k-1}, \rho_1(B_1), \dots, \rho_1(B_m), A_{k+1}, \dots, A_n),$$

where σ_1 is a most general unifier, and we know that

$$\sigma_1(N_0) \wedge (\rho_1 \circ \sigma_1)(C_1) \supset N_1$$

is valid (by lemma 8.5.1). Since ρ_1 is a renaming substitution, it is the identity on N_0 , and by the definition of θ_1 , we have

$$\begin{aligned} & \{\theta_2(\sigma_1(N_0)), \theta_2(\rho_1 \circ \sigma_1(C_1)), \theta_2(C_2), \dots, \theta_p(C_p)\} \\ &= \{\theta_1(N_0), \theta_1(C_1), \theta_2(C_2), \dots, \theta_p(C_p)\}. \end{aligned}$$

If $\{\theta_1(N_0), \theta_1(C_1), \dots, \theta_p(C_p)\}$ was satisfiable, since

$$\sigma_1(N_0) \wedge (\rho_1 \circ \sigma_1)(C_1) \supset N_1$$

is valid,

$$\{\theta_2(N_1), \theta_2(C_2), \dots, \theta_p(C_p)\}$$

would also be satisfiable, a contradiction. Hence,

$$\{\theta_1(N_0), \theta_1(C_1), \dots, \theta_p(C_p)\}$$

is unsatisfiable. \square

Theorem 9.5.1 (Correctness of SLD-resolution as a computational procedure) Let (P, G) be a logic program with query $G = \exists z_1 \dots \exists z_q B$, with $B = (B_1 \vee \dots \vee B_n)$. For every SLD-refutation $\mathcal{R} = \langle N_0, N_1, \dots, N_p \rangle$ for the set of Horn clauses in $P \wedge \neg G$, if \mathcal{R} uses (as in lemma 9.5.3) the list of definite clauses $\langle C_1, \dots, C_p \rangle$, the list of result substitutions (not necessarily ground) $\langle \theta_1, \dots, \theta_p \rangle$, and if $\langle \neg C_{i_1}, \dots, \neg C_{i_k} \rangle$ is the subsequence of $\langle N_0, C_1, \dots, C_p \rangle$ consisting of the clauses in $\{\neg B_1, \dots, \neg B_n\}$ (with $\neg C_0 = N_0$), then

$$\models P \supset \theta_{i_1}(C_{i_1}) \vee \dots \vee \theta_{i_k}(C_{i_k}).$$

Proof: Let P' be the set of formulae obtained by deleting the universal quantifiers from the clauses in P . By lemma 9.5.3, there is a sequence of clauses $\langle N_0, C_1, \dots, C_p \rangle$ from the set $P' \cup \{\neg B_1, \dots, \neg B_n\}$ such that

$$\{\theta_1(N_0), \theta_1(C_1), \dots, \theta_p(C_p)\}$$

is unsatisfiable. But then, it is easy to construct a proof of

$$P \supset \theta_{i_1}(C_{i_1}) \vee \dots \vee \theta_{i_k}(C_{i_k})$$

(using \forall : *right* rules as in lemma 8.5.4), and this yields the result. \square

Note: The formulae C_{i_1}, \dots, C_{i_k} are not necessarily distinct, but the substitutions $\theta_{i_1}, \dots, \theta_{i_k}$ might be.

Corollary Let (P, G) be a logic program such that P is a set of definite clauses and G is a formula of the form $\exists z_1 \dots \exists z_q B$, where B is a conjunction of atomic formulae. For every SLD-refutation of the set of Horn clauses $P \wedge \neg G$, if σ is the result substitution and $(t_1/z_1, \dots, t_q/z_q)$ is any ground substitution such that for every variable z_i in the support of σ , t_i is some ground instance of $\sigma(z_i)$ and otherwise t_i is any arbitrary term in H , then

$$\models P \supset B[t_1/z_1, \dots, t_q/z_q].$$

Proof: First, observe that $\neg B$ must be the goal clause N_0 . Also, if some output variable z_i does not occur in the support of the output substitution σ , this means that $\sigma(z_i) = z_i$. But then, it is immediate by lemma 9.5.3 that the result of substituting arbitrary terms in H for these variables in

$$\{\theta_1(N_0), \theta_1(C_1), \dots, \theta_p(C_p)\}$$

is also unsatisfiable. \square

Theorem 9.5.1 shows that SLD-resolution is a correct method for computing elements of $M(P, G)$, since every set $\{(t_1^1, \dots, t_q^1), \dots, (t_1^k, \dots, t_q^k)\}$ of tuples of terms in H returned by an SLD-refutation (corresponding to the output variables) makes

$$P \supset B[t_1^1/z_1, \dots, t_q^1/z_q] \vee \dots \vee B[t_1^k/z_1, \dots, t_q^k/z_q]$$

valid.

Remark: Normally, we are interested in tuples of terms in H , because we want the answers to be interpretable as *definite* elements of the Herbrand universe. However, by lemma 9.5.3, *indefinite answers* (sets of tuples of terms containing variables) have to be considered. This is illustrated in the next example.

EXAMPLE 9.5.4

Consider the logic program of example 9.5.1. The set of clauses corresponding to $P \wedge \neg G$ is the following:

```

rocksinger(jackson).
teacher(jean).
teacher(susan).
rich(X) : -rocksinger(X).
      : -teacher(X), rich(X).
rocksinger(Y).
      : -rich(Z)

```

Note the two negative clauses. There are four SLD-refutations, two with goal : $-teacher(X), rich(X)$, and two with goal : $-rich(Z)$.

(i) SLD-refutation with output ($jean/Y$):

| Goal clause | Input clause | Substitution |
|--------------------------|----------------------------|--------------|
| : $-teacher(X), rich(X)$ | $teacher(jean)$ | |
| : $-rich(jean)$ | $rich(X) : -rocksinger(X)$ | $(jean/X)$ |
| : $-rocksinger(jean)$ | $rocksinger(Y)$ | $(jean/X_1)$ |
| □ | | $(jean/Y_1)$ |

The result substitution is ($jean/Y, jean/X$). Also, Z is any element of the Herbrand universe.

(ii) SLD-refutation with output ($susan/Y$): Similar to the above.

(iii) SLD-refutation with output ($jackson/Z$):

| Goal clause | Input clause | Substitution |
|----------------------|----------------------------|-----------------|
| : $-rich(Z)$ | $rich(X) : -rocksinger(X)$ | |
| : $-rocksinger(X_1)$ | $rocksinger(jackson)$ | (X_1/Z) |
| □ | | $(jackson/X_1)$ |

Y is any element of the Herbrand universe.

(iv) SLD-refutation with output ($Y_1/Y, Y_1/Z$):

| Goal clause | Input clause | Substitution |
|----------------------|----------------------------|--------------|
| : $-rich(Z)$ | $rich(X) : -rocksinger(X)$ | |
| : $-rocksinger(X_1)$ | $rocksinger(Y)$ | (X_1/Z) |
| □ | | (Y_1/X_1) |

In this last refutation, we have an indefinite answer that says that for any Y_1 in the Herbrand universe, $Y = Y_1, Z = Y_1$ is an answer. This is indeed correct, since the clause $rich(X) : -rocksinger(X)$ is equivalent to $\neg rocksinger(X) \vee rich(X)$, and so

$$\models P \supset (\neg rocksinger(Y_1) \vee rich(Y_1)).$$

We now turn to the completeness of SLD-resolution as a computation procedure.

9.5.4 Completeness of SLD-Resolution as a Computational Procedure

The correctness of SLD-resolution as a computational procedure brings up immediately the question of its completeness. For any set of tuples in $M(P, G)$, is there an SLD-refutation with that answer? This is indeed the case, as shown below. We state and prove the following theorem for the special case of definite clauses, leaving the general case as an exercise.

Theorem 9.5.2 Let (P, G) be a logic program such that P is a set of definite clauses and G is a goal of the form $\exists z_1 \dots \exists z_q B$, where B is a conjunction $B_1 \wedge \dots \wedge B_n$ of atomic formulae. For every tuple $(t_1, \dots, t_q) \in M(P, G)$, there is an SLD-refutation with result substitution σ and a (ground) substitution η such that the restriction of $\sigma \circ \eta$ to z_1, \dots, z_q is $(t_1/z_1, \dots, t_q/z_q)$.

Proof: By definition, $(t_1, \dots, t_q) \in M(P, G)$ iff

$$\begin{aligned} & \models P \supset (B_1 \wedge \dots \wedge B_n)[t_1/z_1, \dots, t_q/z_q] \quad \text{iff} \\ & P \wedge (\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q] \quad \text{is unsatisfiable.} \end{aligned}$$

By theorem 9.5.1, there is an SLD-refutation with output substitution θ_1 . Since

$$(\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q]$$

is the only negative clause, by lemma 9.5.3, for some sequence of clauses $\langle C_1, \dots, C_p \rangle$ such that the universal closure of each clause C_i is in P ,

$$\{\theta_1((\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q]), \theta_1(C_1), \dots, \theta_1(C_p)\}$$

is also unsatisfiable. If θ_1 is not a ground substitution, we can substitute ground terms for the variables and form other ground substitutions $\theta'_1, \dots, \theta'_p$ such that,

$$\{\theta'_1((\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q]), \theta'_1(C_1), \dots, \theta'_p(C_p)\}$$

is still unsatisfiable. Since the terms t_1, \dots, t_q are ground terms,

$$\theta'_1((\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q]) = (\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q].$$

By theorem 9.3.1, there is a ground SLD-refutation \mathcal{R}_g with sequence of input clauses

$$\begin{aligned} & \langle \{(\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q], C'_1, \dots, C'_r\} \quad \text{for} \\ & \{(\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q], \theta'_1(C_1), \dots, \theta'_p(C_p)\}. \end{aligned}$$

By the lifting lemma (lemma 8.5.4), there is an SLD-refutation \mathcal{R} with sequence of input clauses

$$\begin{aligned} & \langle \{\neg B_1, \dots, \neg B_n\}, C''_1, \dots, C''_r \rangle \quad \text{for} \\ & \{\{\neg B_1, \dots, \neg B_n\}, C_1, \dots, C_p\}, \end{aligned}$$

such that for every pair of clauses N_i'' in \mathcal{R} and N_i' in \mathcal{R}_g , $N_i' = \eta_i(N_i'')$, for some ground substitution η_i . Let $\eta = \eta_r$, and let σ be the result substitution of the SLD-refutation \mathcal{R} . It can be shown that

$$(\neg B_1 \vee \dots \vee \neg B_n)[t_1/z_1, \dots, t_q/z_q] = (\sigma \circ \eta)(\neg B_1 \vee \dots \vee \neg B_n),$$

which shows that $(t_1/z_1, \dots, t_q/z_q)$ is equal to the restriction of $\sigma \circ \eta$ to z_1, \dots, z_q . \square

9.5.5 Limitations of PROLOG

Theorem 9.5.1 and theorem 9.5.2 show that SLD-Resolution is a correct and complete procedure for computing the sets of tuples belonging to the meaning of a logic program. From a theoretical point of view, this is very satisfactory. However, from a practical point of view, there is still something missing. Indeed, we still need a procedure for producing SLD-refutations, and if possible, efficiently. It is possible to organize the set all SLD-refutations into a kind of tree (the search space), and the problem is then reduced to a tree traversal. If one wants to retain completeness, the kind of tree traversal chosen must be a breadth-first search, which can be very inefficient. Most implementations of PROLOG sacrifice completeness for efficiency, and adopt a depth-first traversal strategy.

Unfortunately, we do not have the space to consider these interesting issues, but we refer the interested reader to Kowalski, 1979, and to Apt and Van Emden, 1982, where the semantics of logic programming is investigated in terms of fixedpoints.

Another point worth noting is that not all first-order formulae (in Skolem form) can be expressed as Horn clauses. The main limitation is that negative premises are not allowed, in the sense that a formula of the form

$$B : \neg A_1, \dots, A_{i-1}, \neg A, A_{i+1}, \dots, A_n.$$

is not equivalent to any Horn clause (see problem 3.5.9).

This restriction can be somewhat overcome by the *negation by failure* strategy, but one has to be careful in defining the semantics of such programs (see Kowalski, 1979, or Apt and Van Emden, 1982).

PROBLEMS

- 9.5.1.** (a) Give an SLD-resolution and the result substitution for the following set of clauses:

French(Jean).
French(Jacques).
British(Peter).
likewine(X, Y) : -French(X), wine(Y).
likewine(X, Bordeaux) : -British(X).
wine(Burgundy).
wine(Bordeaux).
: -likewine(U, V).

(b) Derive all possible answers to the query *likewine(U, V)*.

- 9.5.2.** Give an SLD-resolution and the result substitution for the following set of clauses:

append(cons(A, L1), L2, cons(A, L3)) : -append(L1, L2, L3).
append(nil, L1, L1).
: -append(cons(a, cons(b, nil)), cons(b, cons(c, nil)), Z)

- 9.5.3.** Give an SLD-resolution and the result substitution for the following set of clauses:

hanoi(N, Output) : -move(a, b, c, N, Output).
move(A, B, C, succ(M), Output) : -move(A, C, B, M, Out1),
move(C, B, A, M, Out2),
append(Out1, cons(to(A, B), Out2), Output).
move(A, B, C, 0, nil).
append(cons(A, L1), L2, cons(A, L3)) : -append(L1, L2, L3).
append(nil, L1, L1).
: -hanoi(succ(succ(succ(0))), Z)

- 9.5.4.** Complete the proof of theorem 9.5.1 by filling in the missing details.

- 9.5.5.** State and prove a generalization of theorem 9.5.2 for the case of arbitrary logic programs.

- * **9.5.6** Given a set S of Horn clauses, an H-tree for S is a tree labeled with substitution instances of atomic formulae in S defined inductively as follows:

(i) A tree whose root is labeled with \mathbf{F} (false), and having n immediate successors labeled with atomic formulae B_1, \dots, B_n , where $: -B_1, \dots, B_n$ is some goal clause in S , is an H-tree.

(ii) If T is an H-tree, for every leaf node u labeled with some atomic formulae X that is not a substitution instance of some atomic formula

B in S (a definite clause consisting of a single atomic formula), if X is unifiable with the lefthand side of any clause $A : -B_1, \dots, B_k$ in S , if σ is a most general unifier of X and A , the tree T' obtained by applying the substitution σ to all nodes in T and adding the k ($k > 0$) immediate successors $\sigma(B_1), \dots, \sigma(B_k)$ to the node u labeled with $\sigma(X) = \sigma(A)$ is an H-tree (if $k = 0$, the tree T becomes the tree T' obtained by applying the substitution σ to all nodes in T . In this case, $\sigma(X)$ is a substitution instance of an axiom.)

An H-tree for S is a proof tree iff all its leaves are labeled with substitution instances of axioms in S (definite clauses consisting of a single atomic formula).

Prove that S is unsatisfiable iff there is some H-tree for S which is a proof tree.

Hint: Use problem 9.2.4 and adapt the lifting lemma.

- * **9.5.7** Complete the proof of theorem 9.5.2 by proving that the substitution $\varphi = (t_1/z_1, \dots, t_q/z_q)$ is equal to the restriction of $\sigma \circ \eta$ to z_1, \dots, z_q .

Hint: Let $\mathcal{R} = \langle N''_0, \dots, N''_r \rangle$ be the nonground SLD-refutation obtained by lifting the ground SLD-refutation $\mathcal{R}_g = \langle N'_0, \dots, N'_r \rangle$, and let $\langle \sigma''_1, \dots, \sigma''_r \rangle$ be the sequence of unifiers associated with \mathcal{R} . Note that $\sigma = \sigma''_1 \circ \dots \circ \sigma''_r$. Prove that there exists a sequence $\langle \eta_0, \dots, \eta_r \rangle$ of ground substitutions, such that, $\eta_0 = \varphi$, and for every i , $1 \leq i \leq r$, $\eta_{i-1} = \lambda_i \circ \eta_i$, where λ_i denotes the restriction of σ''_i to the support of η_{i-1} . Conclude that $\varphi = \lambda_1 \circ \dots \circ \lambda_r \circ \eta_r$.

Notes and Suggestions for Further Reading

The method of SLD-resolution is a special case of the SL-resolution of Kowalski and Kuehner (see Siekman and Wrightson, 1983), itself a derivative of Model Elimination (Loveland, 1978).

To the best of our knowledge, the method used in Sections 9.2 and 9.3 for proving the completeness of SLD-resolution for (propositional) Horn clauses by linearizing a Gentzen proof in SLD-form to an SLD-refutation is original.

For an introduction to logic as a problem-solving tool, the reader is referred to Kowalski, 1979, or Bundy, 1983. Issues about the implementation of PROLOG are discussed in Campbell, 1983. So far, there are only a few articles and texts on the semantic foundations of PROLOG, including Kowalski and Van Emden, 1976; Apt and Van Emden, 1982; and Lloyd, 1984. The results of Section 9.5 for disjunctive goals appear to be original.