L10 Linear Resolution
Chapter 10

1. Motivation

Resolution is an improvement of proof systems (e.g., the one in chapter 7). It was further improved (efficiency purpose) in Chapter 9. We will consider even more efficient methods.

1.1 Linear resolution refutation (proof)

Consider a formula A = {A1, A2, A3, A4}, A1 = {p, q}, A2 = {p, -q}, A3 = {-p, q} and A4 = {-p, -q}.

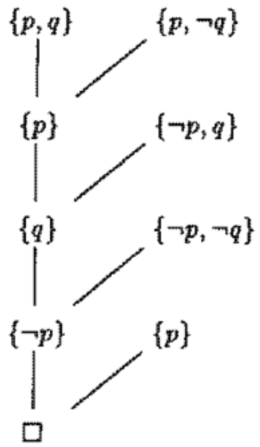

FIGURE 20.

Starting from a clause, we always work on the resolvent, a very strong restriction. It does make the process of finding empty clause easier and more efficiently (i.e., considering less resolutions).

It is no longer complete!

So, we will define a restriction of formula, called **PROLOG program**, for which this method is complete!

Key idea: clause is restricted to **Horn clause**: a clause with at most one positive literal. We have a few types of Horn clauses: program clause (classified into rule, fact), goal clause.
PROLOG program is a set of program clauses.

Remember our goal is to find proofs for $\Sigma \models \sigma$. Different from focusing resolution refutation for formulas, a PROLOG program in fact is always satisfiable. A PROLOG program corresponds to Σ. We know that $\Sigma \wedge \neg\sigma$ is unsatisfiable.

Consider the following propositions as PROLOG program (i.e., Σ). Note we use a more meaningful string to represent a propositional letter

father(john, peter) {father(john, peter)}
father(bill, john) {father(bill, john)}
father(bill, john) → son(john, bill) -father(bill, john) \/ son(john, bill)
{-father(bill, john), son(john, bill)}

father(bill, john) /\ father(john, peter) → grandfather(bill, peter)
-father(bill, john) \/ -father(john, peter) \/ grandfather(bill, peter)
{-father(bill, john), -father(john, peter), grandfather(bill, peter)}

We want to see if proposition σ: grandfather(bill, peter) /\ son(john, bill) is a consequence of Σ.

We known if σ is a consequence of Σ, then $\Sigma \cup \neg\sigma$ is unsatisfiable. $\neg\sigma$ is -(grandfather(bill, peter) /\ son(john, bill)) which is

-grandfather(bill, peter) \/ -son(john, bill)

which is

{-grandfather(bill, peter) \/ -son(john, bill)}

So, we try to find linear resolution refutation of horn clauses

$$\Sigma \cup \{\{\neg grandfather(bill, peter), \neg son(john, bill)\}\}$$

1.2 Linear input (LI) resolution refutation

Given a (PROLOG) program P and a Goal, an LI resolution refutation of $P \cup Goal$ ($$P \cup Goal$$) is a linear resolution refutation with *"starting clause" being Goal* and all "side" clauses are from P.

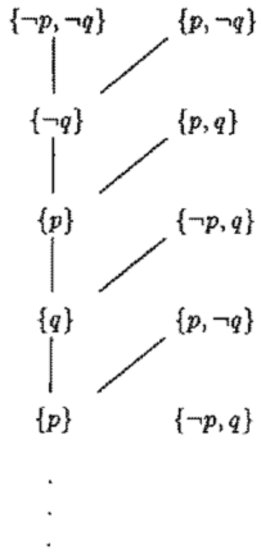

FIGURE 21.

This method is NOT complete for general clauses and a goal!

Lemma 10.5 If P is a PROLOG program and $G = \{\neg q_1, \neg q_2, ..., \neg q_n\}$ a goal clause, then all of the q_i are consequences of P if and only if $P \cup \{G\}$ is unsatisfiable.

Theorem 10.6 If S is an unsatisfiable set of Horn clauses, then there is a linear resolution deduction of $\square$ from S, i.e., $\square \in \mathcal{L}(S)$.

Definition 10.1 (i) A linear (resolution) deduction or proof of C from S is a sequence of pairs $\langle C_0, B_0\rangle, ..., \langle C_n.B_n\rangle$ such that $C = C_{n+1}$ and

(1) C_0 and each B_i are elements of S or some C_j with $j < i$,

(2) each $C_{i+l},\ i \leq n$, is a resolvent of C_i, and B_i.

(ii) As usual we say that C is linearly deducible (or provable) from S, $S \vdash_{\mathcal{L}} C$, if there is a linear deduction of C from S. There is a linear refutation of S if $S \vdash_{\mathcal{L}} \square$. $\mathcal{L}(S)$ is the set of all clauses linearly deducible from S.

Definition 10.2 In the context of linear resolution, the elements of the set S from which we are making our deductions are frequently called input clauses. The C_i are called center clauses and the B_i side clauses. C_0 is called the starting clause of the deduction.

Definition 10.3 (i) A Horn clause is a clause that contains at most one positive literal.

(ii) A program clause is one that contains exactly one positive literal. (In PROLOG notation it looks like $A :- B_1, B_2, ..., B_n.$)

(iii) If a program clause contains some negative literals it is called a rule ($n > 0$ in the notation of (ii)).

(iv) A fact (or unit clause) is one that consists of exactly one positive literal (Notation: $A.$ or $A :-.$).

(v) A goal clause is one that contains no positive literals. (Thus, in PROLOG it is entered as a question with the symbol $?-.$)

(vi) A PROLOG program is a set of clauses containing only program clauses (rules or facts).

1.3 LD-resolution refutation

Change resolution definition: each clause is an ordered tuple and the resolvent clause follows a specific order.

ing procedure on sequences. In particular, when $G = \{\neg A_0, \neg A_1, \ldots, \neg A_n\}$ and $H = \{B, \neg B_0, \ldots, \neg B_m\}$ (viewed as ordered clauses) are resolved, say on $A_i = \neg B$, the interpreter simply replaces A_i by $\neg B_0, \ldots, \neg B_m$. The resolvent is then (as an ordered clause) $\{\neg A_0, \neg A_1, \ldots, \neg A_{i-1}, \neg B_0, \ldots, \neg B_m, \neg A_{i+1}, \ldots, \neg A_n\}$.

G = {-A_0, …, -A_{i-1}, -A_i, -A_{i+1}, …, -A_n}
H = {B, -B_0, …, -B_m}

A_0 /\ …. /\ A_{i-1} /\ A_i /\ A_{i+1}… /\ A_n
B ← B_0, …, B_m

P = {{B}, {-B, A}} Goal: {-A}
{-A}
{A, -B}

{-B}
{B}
{}

Given P = { --> B, B → A} we prove A
A
A ← B

B
B←

[]

Definition 10.10 Let $P \cup \{G\}$ be a set of program clauses, then an LD-resolution refutation of $P \cup \{G\}$ is a sequence $\langle G_0, C_0\rangle, ..., \langle G_n, C_n\rangle$ of ordered clauses G_i, C_i such that $G_0 = G$, $G_{n+1} = \square$, and

(i) Each $G_i, i \leq n$, is an ordered goal clause $\{\neg A_{i,0}, ..., \neg A_{i,n(i)}\}$ of length $n(i) + 1$.

(ii) Each $C_i = \{B_i, \neg B_{i,0}, ..., \neg B_{i,m(i)}\}$ is an ordered program clause of length $m(i) + 2$ from P. (We include the possibility that $C_i = \{B_i\}$, i.e., $m(i) = -1$.)

(iii) For each $i < n$, there is a resolution of G_i and C_i as ordered clauses with resolvent the ordered clause G_{i+1} (of length $n(i) + m(i) + 1$) given by $\{\neg A_{i,0}, ..., \neg A_{i,k-1}, \neg B_{i,0}, ..., \neg B_{i,m(i)}, \neg A_{i,k+1}, ..., \neg A_{i,n(i)}\}$. (In this resolution we resolve on $B_i = A_{i,k}$.)

Lemma 10.11 If $P \cup \{G\} \in$ UNSAT, then there is an LD-resolution refutation of $P \cup \{G\}$ starting with G.

1.4 SLD-resolution refutation

A rule tells us which literal in a resolvent will be further resolved.

Definition 10.12 An SLD-resolution refutation of $P \cup \{G\}$ via (the selection rule) R is an LD-resolution proof $\langle G_0, C_0\rangle, ..., \langle G_n, C_n\rangle$ with $G_0 = G$ and $G_{n+1} = \square$ in which $R(G_i)$ is the literal resolved on at the $(i+1)$ step of the proof. (If no R is mentioned we assume that the standard one of choosing the leftmost literal is intended.)

Theorem 10.13 If $P \cup \{G\} \in$ `UNSAT` and R is any selection rule, then there is an SLD-resolution refutation of $P \cup \{G\}$ via R.

1.5 SLD tree

Given a goal and a PROLOG program, we can draw all possible SLD resolution "sequences" … we use a tree, called SLD tree, to represent all these sequences.

Review

A program rule (exactly one positive literals): e.g., {-A, -B, C}
It is equivalent (semantically) to: -A \/ -B \/ C, equivalent to
A /\ B → C, which is written as
C :- A, B (i.e., reverse conclusion and premise and change conjunction to ",")

E.g., what is the clause for (1) below?

Given program below, I want to prove p

- Working backwards
  - We use rule p :- s first
    Not okay, cannot prove t.
    t;
    s; s :- t (s <- t)
    p; p :- s (p <- s)
  - We use rule p :- q, r first
    ok
    r; r
    q, r; q
    p; p :- q, r (p <- q, r)

- Resolution refutation

< {-p}, {-q, -r, p}>, <{-q, -r}, {q}>, <{-r}, {r}>, <□, _>

```
p :- q, r.      (1)
p :- s.         (2)
     q.         (3)
q :- s.         (4)
     r.         (5)
s :- t.         (6)
     s.         (7)
```

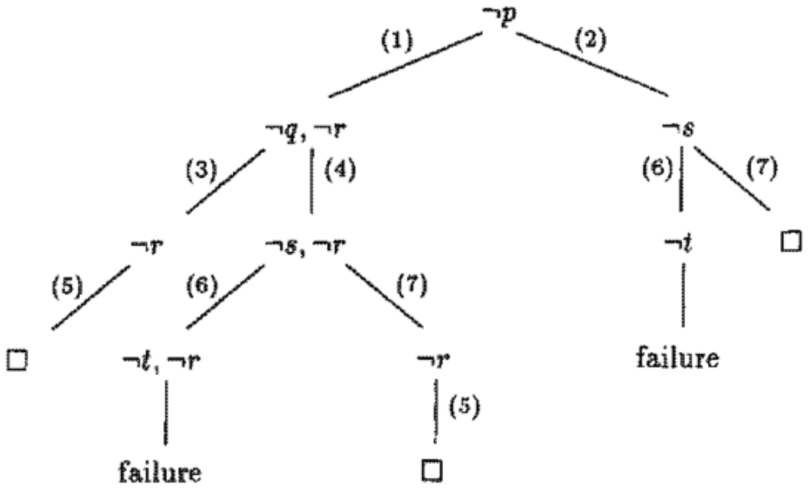

FIGURE 22.