Worksheet 11

Resolution Strategies and Horn Clauses

The aims of this worksheet are that

- you should appreciate something of the size of the search space needed in resolution refutation proofs
- you should see some of the search strategies used to reduce the search space (possibly destroying completeness)

NOTE: A lot of the examples in this chapter are given for the propositional case, i.e. the predicates have no arguments. This is done simply to keep the examples small and tidy. Everything we say also applies in the case of predicates with arguments.

11.1 The Need to Search

Given a set of clauses on which we wish to use resolution, at any stage there are choices:

1. Which clauses do we resolve at each step (i.e. which clauses should be chosen as the parent in the next resolution step)? The strategy that determines this is called the *search rule*.
2. If the chosen clauses (parents) have more than one complementary pair, which literals do we eliminate? The strategy that determines this is called the *selection rule*.

As an example of (1), if the set of clauses is {¬s ∨ ¬r, q ∨ ¬p, p, r ∨ ¬q, s}, we could choose q ∨ ¬p and p as the parents, or we could choose ¬s ∨ ¬r and r ∨ ¬q, and and so on.

As an example of (2), if we have clauses p(a) ∨ p(b) and q(X) ∨ ¬p(X) as parents, we could choose p(a) from the first clause and ¬p(X) from the second as the selected literals, to form the resolvent q(a) ∨ p(b), or we could choose p(b) from the first clause and ¬p(X) from the second, to form the resolvent q(b) ∨ p(a).

Up to now we have presented (resolution) refutation trees, showing the successful resolutions that lie on a derivation of the empty clause from some initial set of clauses. This is misleading. It makes it look as though a human or a machine could unerringly choose the 'right' parents and the 'right' literals within those parents.

In practice, of course, proof is a matter of *search*: trying sequences of resolutions, abandoning them and trying others.

This leads to us distinguishing three notions:

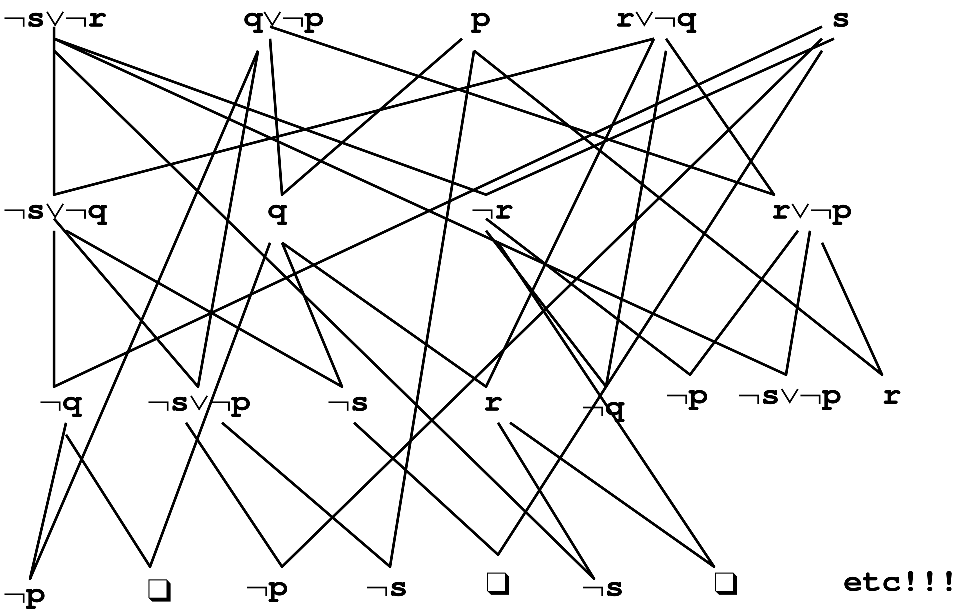
The search space: the set of *all* possible resolvents for a set of clauses;

The derivation graph: the set that we (or the machine) actually try in searching for the empty clause (some of them, of course, being fruitless);

The (resolution) refutation tree: as before, this is the tree within the derivation graph that shows the derivation of the empty clause. (It is also sometimes called the derivation tree or the proof tree.)

Example 1: The clauses are {¬s ∨ ¬r, q ∨ ¬p, p, r ∨ ¬q, s}.

Here is a (partial) search space for the set of clauses labelled example 1. (The reason it's only partial is that there is so much of it!)



You can see that the search space contains several refutation trees.

A machine obviously needs some systematic way of carrying out resolutions in its search for the empty clause. The strategy it uses to decide what to try next is known as its *control strategy*. We will look at a number of control strategies.

11.2 Control Strategies

(The names I give below for different control strategies are as standard as I can manage, but there seems to be little consistency in this area.)

In the *breadth-first strategy* (also called the level saturation strategy), all of the first-level resolvents are computed first, then the second-level resolvents, and so on. On the diagram above, resolvents would be computed layer by layer of the search space. (Obviously, you can stop as soon as you find □).

This strategy is complete but is grossly inefficient.

Example 2: The clauses are $\{\neg p \vee \neg q,\ p \vee \neg q,\ q \vee \neg p,\ p \vee q\}$

Exercise 11.1 *Show the derivation graph down to the third-level resolvents that would be produced by the breadth-first strategy applied to the clauses labelled example 2.*

There are certain ways of improving the efficiency of this, and you might encounter these in your reading.

One possibility is to use so-called *deletion strategies*, which prune useless clauses from the search space. A simple example is that clauses that are tautologies can be removed, e.g. p ∨ q ∨ ¬q ∨ r simplifies to p ∨ `true` ∨ r, which simplifies to `true`, which can be ignored.

Another possibility is to have evaluable predicates. This is to say that certain predicate symbols and function symbols might be given specific interpretations, e.g. the predicate symbol > could have its interpretation fixed to mean 'greater than', the function symbol `+` could have its interpretation fixed to mean addition. If the machine then has procedures for directly computing the greater than relation and the addition function then it can use these to simplify clauses. For example, >`(+(6, 2), 9)` can be evaluated to >`(8, 9)` and further evaluated to `true`. (This crucially depends on > and `+` having fixed interpretations.)

Another possibility is to come up with a different control strategy: one which is more selective and does not compute all resolvents. Many such strategies have been proposed. But clearly, if a control strategy does not compute all resolvents then there is a risk that it may make your system incomplete.

In the strategy called *linear resolution* (also known as ancestry-filtered resolution), the only resolutions computed are those in which at least one of the parents is either a member of the initial clauses or is an ancestor of the other parent. This strategy focuses the search by avoiding resolution of intermediate conclusions with other intermediate conclusions. With an otherwise breadth-first strategy, this strategy is still complete.

If you look at the search space depicted earlier for the clauses of example 1, you will see that certain resolvents would no longer be computed, e.g. this strategy does not bother to form the resolvent of r ∨ ¬p and ¬r since neither of these is an initial axiom and neither is an ancestor of the other.

Then, there is a strategy called *set-of-support resolution*, which is a refinement of linear resolution, in which at least one of the parents is from the negated goal or its descendants. With an otherwise breadth-first strategy, this strategy is still complete.

Exercise 11.2 *Show the whole derivation graph for set-of-support resolution on the clauses of example 1. Take the goal to be ¬s ∨ ¬r. Note how much smaller the explored search space is.*

In *unit resolution* one of the parents is a unit clause, i.e. one consisting of only one literal. While this can be very efficient, it is *not* complete.

Exercise 11.3 *Show the derivation graph of unit resolution on the clauses of example 1. (A refutation tree should be found.) Then show the derivation graph for example 2. (We know from exercise B.0.1 that example 2's clauses are inconsistent, but using unit resolution a refutation will not be found.)*

In *input resolution* at least one of the parents is a member of the initial clauses. This too can be very efficient, but it is *not* complete.

11.3 Horn Clauses

We have seen that many of the more efficient control strategies (e.g. input resolution and unit resolution) are not complete. It turns out that they are only complete provided your initial clauses satisfy certain properties: if your initial clauses satisfy these properties you can use an efficient control strategy; if not, too bad: you'll have to use one of the less efficient strategies.

Specifically, input resolution and unit resolution are complete if all the initial clauses are *Horn clauses.*

A Horn clause is a clause with at most one positive literal.

Examples of Horn clauses and clauses that are non-Horn[1]:

Horn clauses
$\square$
$\neg q$
$\neg q \lor \neg r$
$\neg q \lor \neg r \lor \neg s$
p
$p \lor \neg q$
$p \lor \neg q \lor \neg r$
$p \lor \neg q \lor \neg r \lor \neg s$

Non-Horn clauses
$p \lor q$
$p \lor q \lor \neg r$
$p \lor q \lor r$
$p \lor q \lor r \lor \neg s$
$p \lor q \lor r \lor s$

(Remember that, for now, the order of the literals in a clause is immaterial.)

Horn clauses themselves are further subdivided into

- positive Horn clauses: those with exactly one positive literal
  (also known as Definite Clauses, Kowalski Clauses, Headed Horn clauses)
- negative Horn clauses: those with no positive literals
  (also known as Indefinite Horn clauses, Headless Horn clauses)

The Horn clauses from the table above subdivide as follows:

positive Horn clauses	negative Horn clauses
p	$\square$
$p \lor \neg q$	$\neg q$
$p \lor \neg q \lor \neg r$	$\neg q \lor \neg r$
$p \lor \neg q \lor \neg r \lor \neg s$	$\neg q \lor \neg r \lor \neg s$

[1] As elsewhere, the examples could equally well show cases where the predicates have arguments.

Exercise 11.4 *For all of the clauses from examples 1 and 2 used earlier, say which are Horn clauses, and which are not. For those that are, say whether they are positive or negative Horn clauses.*

Thus confirm that the theorem-proving example from above that worked using unit resolution contained only Horn clauses, and that the other example included one or more non-Horn clauses.

If our initial clauses are Horn clauses, then there is another efficient method that can be used in place of unit resolution or input resolution. It is called SLD-resolution. (This is what Prolog uses.)

11.4 SLD-Resolution

No one seems to agree what SLD stands for (Select Literal resolution on Definite Clauses/ Linear resolution with Selection rule on Definite clauses). Either way, they all agree on what resolutions get computed.

Remember that SLD-Resolution is complete only for Horn clauses.

We are going to look at examples in which there is only one negative Horn clause and any number of positive Horn clauses. There is no loss of generality in this because any solvable resolution refutation problem that can be expressed in Horn clauses (i.e. any theorem-proving problem for which a refutation exists which can be expressed as Horn clauses) can be expressed in such a way that:

- there is only one negative Horn clause, and
- all the rest are positive Horn clauses.

The proof of this is beyond this course.

The negative clause will be thought of as the goal.

In the positive Horn clauses there is exactly one positive literal. It is conventional to write this positive literal leftmost in the clause (as was done above). This does have an effect on the SLD strategy.

Set of support

SLD involves using linear resolution, specifically the modification of linear resolution known as set of support resolution. In this, one of the parents is from the negated goal or one of its descendants.

Input resolution

You will recall that in input resolution one of the parents is always a member of the initial clauses. This too is part of SLD.

Set of support + Input = Linear input resolution

SLD uses set of support to choose one parent and input resolution to choose the other. Together this gives linear input resolution: one parent is from the negated goal or one of its descendants; the other is from the initial clauses.

Note that the parent from the initial clauses will have a positive literal. And the other parent, being the negated goal or one of its descendants, will be all negative.

Ordered resolution (the selection rule)

As mentioned before, a selection rule is needed to decide which complementary pair to use to form a resolvent, i.e. to pick the literals from within the parents. In SLD Resolution this is done by making use of an ordering on clauses.

We only resolve on the *leftmost* literals in *both* clauses.

We know already that positive clauses are ordered with the positive literal leftmost. Thus we need only look at the first literal in the negated goal or its descendant and the first literal in each member of the initial clauses to see what resolvent can be formed.

Thus, typically, we are resolving Horn clauses of the form

$P \vee \neg Q1 \vee \ldots \vee \neg Qi$
and
$\neg P \vee \neg R1 \vee \ldots \vee \neg Ri$.

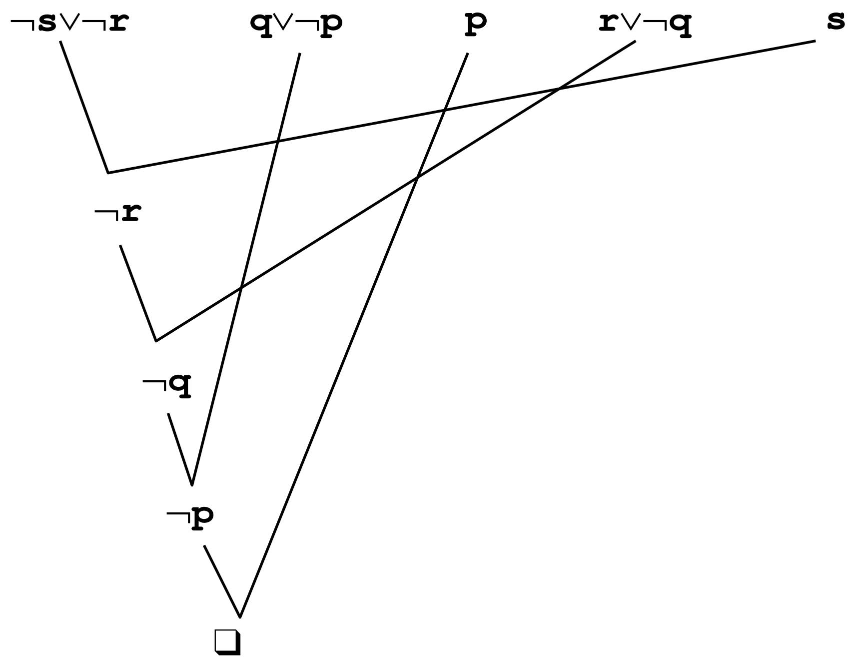
Before I can say what the resolvent is though I have to stipulate that the resolvent too preserves the order from its parent clauses with the literals from the positive parent followed by the literals from the negative parent. This gives: $\neg Q1 \vee \ldots \vee \neg Qi \vee \neg R1 \vee \ldots \vee \neg Ri$.

(This, of course, is a descendant of the goal. It is therefore eligible to be a parent in another resolution step. We would therefore look for another parent from the initial axioms, which would be a positive clause beginning with $Q1$.)

SLD

This finishes the definition of SLD resolution. To recap: at any step, one parent is the goal or one of its descendants. The other parent is from the initial clauses. We resolve on leftmost literals. But all initial clauses are ordered (their positive literal first) and all new resolvents are ordered (their positive parent stuff before their negative parent stuff, otherwise same order as parents).

Here is the SLD-resolution derivation graph for the Horn clause example (example 1), given again that ¬s ∨ ¬r is the goal.



Exercise 11.5 *Give the SLD-resolution derivation graph for the following clauses* $\{\neg r,\ r \vee \neg p \vee \neg q,\ p \vee \neg s \vee \neg t,\ p \vee \neg u,\ q,\ s,\ u\}$*, given that* $\neg r$ *is the goal.*

11.5 Conclusions

It turns out that the restricted theorem-proving we've introduced here (SLD resolution refutation on Horn clauses in which the goal is a negative Horn clause and all the other initial clauses are positive Horn clauses) is powerful enough for us to write programs/formulate theorem-proving problems of as much generality as is needed. In other words, any Turing-machine-computable function can be expressed as a theorem-proving problem of this kind.

Acknowledgements

Genesereth & Nilsson were among the clearer sources consulted when writing this worksheet.