Mark E. Stickel

A Prolog Technology Theorem Prover:

Implementation by an Extended Prolog Compiler

Journal of Automated Reasoning

vol. 4, pp. 353-380, 1988

Discussion led by John Harrison

University of Cambridge

- PTTP: history and its place in ATP
- Horn clauses and Prolog
- From Prolog to PTTP
- Refinements

Model elimination

The deductive procedure underlying PTTP is Donald Loveland's MESON model elimination method, which was invented in the sixties.

Model elimination is described by Loveland in JACM vol. 15 (1968), pp. 236-251 and MESON is described in his 1978 book: 'Automated Theorem Proving: A Logical Basis' (North-Holland).

ME was developed before Loveland had heard of resolution. Loveland's later development of linear resolution was quite separate.

ME is a general proof method for first order logic, and does not (directly) support equality reasoning, arithmetic etc.

PTTP

The idea underlying Stickel's PTTP was to implement the MESON procedure using 'Prolog Technology'.

That is, he made just a few small modifications to a standard Prolog system (details later) and obtained a system complete for first order logic.

It's probably thanks to PTTP that model elimination didn't disappear completely against the background of the intense interest in resolution.

SETHEO (from Munich), winner of the 1996 CADE theorem proving competition, is basically a well-engineered version of PTTP.

The second-placed system, Otter, is the current resolution flagship.

There are implementations of similar algorithms in Isabelle (`meson_tac`) and in HOL (`MESON_TAC`), though here clauses are *interpreted* not *compiled*.

Where ME belongs

We can divide the standard first order theorem proving methods into two main groups:

- The bottom-up, 'local' methods, e.g. resolution (Robinson, JACM 1965) and the inverse method (Maslov, Dok. Akad. Nauk 1964).
- The top-down, 'global' methods, e.g. model elimination and tableaux.

In some sense, *all* these can be seen as search for a proof in cut-free sequent calculus, using unification to discover instantiations for quantifiers.

The bottom-up methods start at the assumptions and deduce an ever-increasing set of facts till they reach the conclusion. Top-down method work backwards from the conclusion, breaking it down to subproblems until the assumptions are reached.

Top-down vs. bottom-up

The bottom-up methods have several advantages. Effectively they perform proof at the meta-level: we can regard free variables as implicitly universally quantified.

Therefore it is possible to apply subsumption to the current set of facts, and avoid proving the same lemma twice. By contrast, in top-down ('global') methods, the free variables in different subgoals need to be correlated.

However, top-down methods are more goal-directed: we don't just grow a big set of facts and hope we reach the conclusion.

Moreover, they are much more economical to implement, since we only need to store the current subgoals. In fact, they are *all* very Prolog-like: apart from the PTTP implementation of MESON, there is a complete tableau prover called leanT^AP that requires only 5 lines of Prolog.

leanT^AP

This is due to Beckert and Posegga; see the Journal of Automated Reasoning, vol. 15, pp. 339-358, 1995.

```
prove((E,F),A,B,C,D) :- !,prove(E,[F|A],B,C,D).
prove((E;F),A,B,C,D) :- !,prove(E,A,B,C,D),
                          prove(F,A,B,C,D).
prove(all(I,J),A,B,C,D) :- !,
   \+length(C,D),copy_term((I,J,C),(G,F,C)),
   append(A,[all(I,J)],E),prove(F,E,B,[G|C],D).
prove(A,_,[C|D],_,_) :-
    ((A= -(B);-(A)=B) ->
     (unify(B,C);prove(A,[],D,_,_))).
prove(A,[E|F],B,C,D) :- prove(E,F,[A|B],C,D).
```

This sort of naive tableau prover is the core of Isabelle's `fast_tac` and HOL's `TAB_TAC`.

Horn clauses and Prolog

A *clause* is a disjunction of literals, where a literal is either an atomic formula or its negation:

$$L_1 \vee \cdots \vee L_n$$

We say it is a *Horn clause* if it has at most one unnegated literal. In this case we can write it as

$$-L_1 \wedge \cdots \wedge -L_{k-1} \wedge -L_{k+1} \wedge \cdots \wedge -L_n \Longrightarrow L_k$$

or simply 'L_1' if $n = 1$. These are the clauses that are allowed in a Prolog database. The Prolog syntax for the prototypical Horn clause is:

$$L_k \texttt{:-} - L_1, \ldots, -L_{k-1}, -L_{k+1}, \ldots, -L_n$$

Prolog allows us to deduce an atomic formula from such a database by backchaining through the rules, using unification to instantiate variables (written in upper case in Prolog).

Why is Prolog inadequate?

Prolog certainly has a limited ability to prove theorems. However it is inadequate as a general first order prover for three reasons:

- Most Prolog implementations have unsound unification
- Prolog is limited to Horn clauses
- Prolog's depth-first search strategy is incomplete.

We arrive at PTTP by fixing each of these problems. We will consider them in turn.

Unsound unification

It has long been usual for Prolog implementations to omit the so-called 'occurs check', e.g. allowing X and $f(X)$ to be unified.

This is either for (probably bogus) efficiency reasons, or because circular data structures are sometimes considered useful.

However it's disastrous for theorem proving, e.g. it would allow us to deduce `SUC(Y) < Y` from `X < SUC(X)`.

The fix is easy: just do unification properly.

Limitation to Horn clauses

It is not always possible to reduce theorem proving problems to Horn clause sets acceptable to Prolog.

For example, we might want to use the facts $A \vee B$ and $A \Longrightarrow B$ to deduce B. However there is no equivalent in terms of Horn clauses.

The solution adopted in PTTP is to extend the notion of 'Horn clause':

$$A_1 \wedge \cdots A_n \Longrightarrow B$$

to allow any or all of the literals involved to be negated.

Now we can take any problem and reduce it to something based on these pseudo-Horn clauses.

Contrapositives

We take the fact we want to prove (maybe an implication under a set of assumptions), negate it, Skolemize it and reduce it to clausal form. We want to derive $\bot$. For each clause:

$$P_1 \vee \ldots \vee P_n$$

we form n *contrapositives* of the form:

$$-P_1 \wedge \cdots \wedge -P_{i-1} \wedge -P_{i+1} \wedge \cdots \wedge -P_n \Longrightarrow P_i$$

and one more of the form:

$$-P_1 \wedge \ldots \wedge -P_n \Longrightarrow \bot$$

Now we try to solve the goal $\bot$ à la Prolog.

Incompleteness

Unfortunately, while Prolog-style backchaining is complete for true Horn clauses, this is not so for pseudo-Horn clauses. Consider the intended example of deducing B from $A \vee B$ and $A \Longrightarrow B$. The contrapositives are:

$$\begin{array}{rcl} B & \Longrightarrow & \bot \\ \neg A & \Longrightarrow & B \\ \neg B & \Longrightarrow & A \\ \neg A \wedge \neg B & \Longrightarrow & \bot \\ A & \Longrightarrow & B \\ \neg B & \Longrightarrow & \neg A \\ A \wedge \neg B & \Longrightarrow & \bot \end{array}$$

It is immediate that no Prolog-style search can terminate in success because there are no unit clauses.

Ancestor unification

We can restore completeness by an extra rule: as well as unification with the conclusion of a rule, we allow unification with *the negation of an ancestor*.

This is treated as a unit clause and can solve a goal; note that the variables, if any, are correlated. For example

$$\perp \leftarrow B \leftarrow A \leftarrow \neg B$$

Now we can unify $\neg B$ and the negation of B.

The logical justification is simple: if we are trying to prove a goal, here B, we may assume its negation $\neg B$, since if that is false we are immediately finished.

Search strategy

Although this is now complete *as a calculus*, the usual Prolog depth-first search with rules tried in order is trivially incomplete.

For example, the rules $P(f(X)) \Longrightarrow P(X)$ and $P(f(a))$ cannot solve the goal $P(a)$ because Prolog will keep applying the first rule ad infinitum:

$$P(a) \leftarrow P(f(a)) \leftarrow P(f(f(a))) \leftarrow P(f(f(f(a)))) \cdots$$

We need to use a search strategy that will allow all possible proofs to be found.

The most obvious is breadth-first search. But this blows up the storage requirement: the minimal storage usage is one of PTTP's strong points.

Instead, most implementations use *depth first iterative deepening*: search for proofs of depth 1, then if that fails, depth 2, then if that fails, depth 3, and so on.

Search alternatives

Various search modes have been experimented with

- Loveland originally used iterative deepening where 'depth' is maximum height of proof tree.
- Stickel then used depth = size of proof tree (number of nodes).
- Paulson uses best-first search (not iterative deepening).
- SETHEO allows height or size bounds; the former is better on average.
- Harrison uses an optimized version of size bounds, which seems better still.

Implementation refinements

- Making larger DFID increments automatically (e.g. if all the clauses are units or have 3 assumptions, we can increment by 3 each time).
- Avoiding repetitions down a branch of the proof tree. (Though these may only appear after later instantiation, so there are trade-offs to be made.)
- Performing 'intelligent backtracking', e.g. if a goal is solved using a unit clause or ancestor without instantiation of the goal, no other solutions need be tried. More advanced optimizations can interact badly with iterative deepening.
- Extending the system to nonclausal assertions, à la Prolog, e.g. $P \wedge (Q \vee R) \Longrightarrow S$ instead of two separate clauses. This avoids repeating the solution of P.

Refinements to the calculus

It suffices to generate rules with conclusion $\perp$ only if all the literals in the clause are negated. Often this is just the original goal. Also, there is Plaisted's 'positive refinement'.

There are alternative versions of ME that only use 'natural' contrapositives, e.g. Loveland's *Near-Horn Prolog*, Plaisted's *Modified Problem Reduction Format* and Baumgartner & Furbach's *Restart Model Elimination*.

There are also various techniques for caching and lemmatizing. These were originally used by Loveland, and fell out of favour, but are now attracting attention again. For example, SETHEO uses several quite sophisticated techniques.