Automated Deduction and Algebra
Lecture II

Michael K. Kinyon

Department of Mathematics

University of Denver

University of Lisbon, 9 June 2009

Order?

Beginners often make two *incorrect* assumptions when first working with Prover9 or Mace4:

- The variables range over some domain, such as the real numbers.
- Symbols have their usual meanings.

Mathematicians have no trouble ridding themselves of the first assumption.

The second, however...

Binary relations

Besides unary predicates, which we looked at last time, Prover9 can handle general n-ary relations. Here is an example from semigroup theory.

Surprising Behavior

When reading through output, one often finds a clause which seems promising in one of two ways:

- It is obvious to an expert, OR
- It looks important enough to warrant its own lemma.

In either case, the temptation is to try including the clause among the assumptions to see if this will help find a proof or simplify an existing proof.

The problem is that this doesn’t always work. . .

Helping Mace4

By contrast, model builders such as Mace4 usually work better with *more* assumptions.

However, there is a trade-off; clauses with many variables can slow Mace4 down considerably because of the necessary evaluations.

Restricting denials

Sometimes Prover9 spends too much time reasoning backwards from the denials, often in trivial ways. We saw an example of this yesterday. A way of preventing this is to use

```
set(restrict_denials).
```

The effect of this is that paramodulation and resolution will not be applied to denials. They will, however, still be back rewritten.

We'll see an example of this in a moment.

Setting the flag can speed up proofs, but it can also slow down proofs and even block proofs. Use with caution.

The Robbins Problem

This is the problem solved by McCune in 1996:
Do the following equations axiomatize Boolean algebras?

```
x + y = y + x                    # label(Commutativity
(x + y) + z = x + (y + z)        # label(Associativity
((x + y)' + (x + y')')' = x      # label(Robbins).
```

The following condition was known for a long time to be sufficient:

```
exists a exists b (a + b)' = a' # answer(Winker).
```

Using definitions

McCune used EQP, a relative of OTTER, in 1996 to solve the problem. It took 7.85 *days* of CPU time.

Using cleverly chosen definitions, it's easier now...

Sensitivity to function order

Some problems can be very sensitive to `function_order`.

Even with the `eq_defs` flag set, things still might not go well...

Labels are exactly what they sound like. They just provide a useful way of labeling clauses.

Answer is not only a label, it also sends a message to standard output when a proof is found. This can be very useful, because it can output the final value of a variable...

Too much symmetry

Some problems possess too much symmetry. Prover9 can easily get lost in the search space because of rearranging symmetric functions in essentially trivial ways.
This problem is well-known in computer science for commutative-associative operations. Here is another example:

```
f(x,x,y) = x.
f(x,y,z) = f(z,x,y).
f(x,y,z) = f(x,z,y).
f(f(x,w,y),w,z) = f(x,w,f(y,w,z)).
```

These axioms define *median algebras*.

The problem

The problem was to find a first order proof of the following "distributive law":

```
f(f(x,y,z),u,w) = f(f(x,u,w),f(y,u,w),f(z,u,w)).
```

Prior to 1995, the only proof of this was high order.

The problem was posed to McCune and Bob Veroff by Donald Knuth. The key to finding a proof was to cut the symmetry involved...

lex order vars

The idea is to put an ordering on variables:

```
set(lex_order_vars).
```

The effect of this is to treat variables as constants when comparing terms, with the precedence

```
function_order([x,y,z,u,w,v5,v6,v7, ...]).
```

This effectively breaks the symmetry. For instance, the commutative law `x * y = y * x.` becomes orientable.

Caveat

It's worth a try and can drastically reduce the search space.

But it's an incomplete method, and can therefore easily block proofs.

New constants

Sometimes a theorem prover will derive an equation in which each side contains only variables and those variables are distinct, *e.g.*,

```
x' ^ x = y ^ y'.
```

We recognize that this means both expressions are constant. Prover9 can do this too with the following option:

```
assign(new_constants,1).
```

(This can also be set higher.)

prooftrans

As we have already noted, reading a Prover9 proof can sometimes be rather difficult. For instance, if a justification has a lot of rewrite steps, this can make the proof hard to follow.

There is a way of expanding proofs, however, that gets rid of rewrites. . .

op

As came up yesterday, you might want to avoid excessive parenthesization in associative operations. This is easy to do by redeclaring the parsing type, e.g.,

```
op(450, infix_right, "*").
```

You can define your own symbols for operations:

```
op(400, infix, "o")
```

You can also change some default symbols, such as logical connectives:

```
redeclare(conjunction, AND).
```

Proof sketches

Now we will discuss a very powerful advanced technique called *proof sketches*, designed by Bob Veroff.

The basic idea: suppose the theorem we want to prove is easier to prove in a more specialized setting. It might nevertheless be the case that the proofs of both the special and general cases have features in common.

It would be nice if we could “guide” Prover9 in the direction of the proof of the special case as it attempts to prove the general case.

Theoretical example

To make this concrete, suppose you want to prove a theorem about lattices, and the theorem is easy to prove in Boolean algebras.
One first might try proving it in modular ortholattices. If that works, then try orthomodular lattices, then try ortholattices, then complemented lattices, and finally, lattices themselves. Each time, we use the previous proof to “guide” the current search.

The idea

Consider a proof as a sequence of clauses:

$$C_1, C_2, \ldots, C_i \ldots, C_j \ldots, C_n$$

where

- C_i is an extra assumption in the target theory
- C_j has C_i in its derivation history.

Now C_j is either in the target theory or not.

- If yes, it is enough to find a new proof of C_j.
- If no, it is enough to "bridge the gap".

In either case, we've reduced a large problem to a hopefully smaller one.

Implementation

Recall that in Prover9, “it’s all about the given clause”.

Besides cycling through old clauses, light clauses, and so on, a part of Prover9’s given selection cycle is given over to selecting clauses called *hint matchers*.

A list of *hints* is just a set of clauses. If a derived clause subsumes a hint, it is said to *match* the hint.

By default, whenever hint matchers are available for selection as given clauses, Prover9 will use them immediately.
Hint matchers are “notable milestones” toward a proof.

By the way…

By default, hints are rewritten and simplified with respect to whatever the current set of demodulators is.

The reason is that otherwise a clause may fail to match a hint because the clause was rewritten while the hint was not.

Where to get good sketches?

Good proof sketches can be obtained in various ways.

- Prove theorem in a stronger theory (*e.g.*, lattice hierarchy)
- Assume generalizations (*e.g.*, by renaming variables)
- Prove special instances of theorems (*e.g.*, by renaming variables)
- Replace assumptions by proper instances (as part of the process of eliminating assumptions)

A practical issue

If you have a lot of hints in separate lists, the GUI can become a bit of a pain. At that point, you will want to learn how to use the command line versions of Prover9 and Mace4.