COMMUTATIVE A-LOOPS II: THE AUTOMATED DEDUCTION SIDE

Michael K. Kinyon

Department of Mathematics

ADAM 2008

Definition

```
% commutative loop
0 * x = x.
x * y = y * x.
x \ (x * y) = y. x * (x \ y) = y.

% left inner mapping
(x * y) \ (x * (y * z)) = L(x,y,z).

% automorphism
L(x,y,z) * L(x,y,u) = L(x,y,z * u).
L(x,y,z) \ L(x,y,u) = L(x,y,z \ u).
```

Square roots

While we were all riding in Petr's car, Petr and Přemysl asked me if the product of squares is a square. Here is how I set it up:

```
(A * A) * (B * B) != x * x # answer(x).
```

PROVER9 gets a proof rather quickly.

Square roots

While we were all riding in Petr's car, Petr and Přemysl asked me if the product of squares is a square. Here is how I set it up:

```
(A * A) * (B * B) != x * x # answer(x).
```

PROVER9 gets a proof rather quickly. The next issue was to get

x into a reasonable form. I did this by padding the input with a couple of axioms.

Derived operation

As Petr discussed in his talk, we decided to treat the expression we obtained as a new binary operation $+$, so that

$$x^2y^2 = (x + y)^2$$

For a loop in which squaring is a permutation, this just gives an isomorphic copy of the loop you start with.

In general, it turns out this new operation is commutative:

$$x + y = y + x\,.$$

The PROVER9 proof of this was not too bad to translate.

Exponent 2 case

Here is what the new operation looks like in the exponent 2 case:

```
% derived operation
y \ ((x * y) \ y) = x + y.
```

The first thing we found, which turned out later to be easy to show by hand, is that $+$ gives a Steiner loop:

```
x + y = y + x.
x + (y + x) = y.
```

So the question was: is $+$ associative?

```
formulas(goals).
(x + y) + z = x + (y + z) # label("assoc").
end_of_list.
```

Setting It Up

My personal default term ordering is KBO:

```
assign(order,kbo).
```

While LPO is occasionally faster, in loop theory problems, it usually generates less comprehensible proofs.

Setting It Up

My personal default term ordering is KBO:

```
assign(order,kbo).
```

While LPO is occasionally faster, in loop theory problems, it usually generates less comprehensible proofs.

Because the problem has two commutative operations in it, left to its own devices, PROVER9 will often waste time generating trivial consequences of commutativity. So I used

```
set(lex_order_vars).
```

This is a symmetry-breaking trick. It is known to be incomplete, and can block proofs.

Semantic Guidance I

I used semantic guidance for this problem. First, I used MACE4 to generate a model in which the A-loop axiom was false. I kept the rest of the hypotheses, including $+$ being Steiner.

By the way, contra the manual, I did not (and usually do not) use the goal as part of the MACE4 search. I want PROVER9 to pay more attention to consequences of the key hypothesis; I don't care if the goal itself is or is not true in the model. (After all, the key hypothesis might be sufficient, but not necessary.)

Semantic Guidance II

The smallest nonassociative Steiner loop has order 10, so I also needed:

```
assign(eval_limit,10000).
```

Semantic Guidance II

The smallest nonassociative Steiner loop has order 10, so I also needed:

```
assign(eval_limit,10000).
```

I really didn't get anywhere with all of this for several days.

Semantic Guidance III

Looking at the output, it finally dawned on me what the problem was: the false clauses PROVER9 was generating were mostly trivial, involving only two variables.

Semantic Guidance III

Looking at the output, it finally dawned on me what the problem was: the false clauses PROVER9 was generating were mostly trivial, involving only two variables.

Here is the key hypothesis again:

```
% automorphism
L(x,y,z) * L(x,y,u) = L(x,y,z * u).
```

This has four variables. So I reran the MACE4 search, this time assuming that a particular *instance* of the hypothesis is still true:

```
L(x,y,x) * L(x,y,y) = L(x,y,x * y).
```

Semantic Guidance IV

I think this is a nice general method when using semantic guidance:

Find a model in which not only is the key hypothesis false, but particular instances of it are true.

(Of course, a model might be too big to be useful.)

One more thing . . .

In these first attempts, I was *not* using the function

```
% left inner mapping
(x * y) \ (x * (y * z)) = L(x,y,z).
```

as part of the input. Instead, everything was expressed in terms of $*$ and $\backslash$.

(I had no special reason for doing it this way.)

Success!

```
% Proof 1 at 40328.23 (+ 406.18) seconds
% Length of proof is 147.
% Level of proof is 31.
% Maximum clause weight is 39.
% Given clauses 3343.
```

The proof is not very long, but I found it very difficult to parse, *e.g.*,

```
15838 (x \ y) * ((x \ (z \ y)) * (((x \ (z \ y))
\ (x \ z)) \ (x \ y))) = z * (((x \ z) * (x \
(z \ y))) \ (x \ y)) # label(false).
```

Reintroducing defined functions

So the first thing I did was rerun the job, using the first proof for hints, but this time reintroducing the function

```
% left inner mapping
(x * y) \ (x * (y * z)) = L(x,y,z).
```

I put it midway in

```
function_order([0,+,*,L,\]).
```

(This matters more with LPO than it does with KBO.)

Simple Proofs

Simple proofs are easier to translate into human form. So it's worth spending a bit of time simplifying a proof before beginning to translate it.

Crude measures of proof complexity:

- the length of a proof as reported by Prover9
- the level of the proof
- the maximum clause weight in the proof
- the length of the expanded proof after it has passed through Prooftrans.

Getting a stable proof I

The proof for this second job did not take too long (about 7 or 8 minutes).

Length 270, Level 50, Max weight 36, Given 3911, Expanded length 732

This is longer than what we started with, but we've only begun.

This proof is not *stable*, that is, PROVER9 does not reproduce the same proof by following the hints generated from the proof. The hints are dropped at given 177.

Getting a stable proof II

The first thing to do is to rerun the job with the original proof as hints and no other changes to the input:

Length 241, Level 50, Max wt 28, Given 1775, Exp length 594

This is still not stable, so we try repeating it.

But in this case, there was no proof after 4000 givens!

So we repeat using the hints from both of the previous jobs.

Length 210, Level 45, Max wt 35, Given 389, Exp length 510

Getting a stable proof III

Repeat, using just the hints from the previous job. This time, Prover9 does not lose the hints.

Length 148, Level 54, Max wt 31, Given 148, Exp length 332

Repeat:

Length 147, Level 47, Max wt 28, Given 119, Exp length 337

This proof is stable. On the next run, PROVER9 reproduces it perfectly after 119 givens.

Squeezing the max weight I

One of the easiest ways to simplify a proof is try to reduce the maximum clause weight in the proof. This forces Prover9 to find different proofs of steps, and these will most often be shorter than what we had before.

```
set(limit_hint_matchers).
assign(max_weight,28).
```

Not surprisingly, this just reproduced the stable proof. So we try:

```
assign(max_weight,27).
```

Squeezing the max weight II

Length 147, Level 47, Max wt 25, Given 119, Exp length 340

It is almost the same proof. It differs by a just a few clauses.

Next, we reduce max weight to 25 and repeat.

Length 153, Level 44, Max wt 25, Given 119, Exp length 357

The length has gone up but the level has dropped, so let us stick with it for a while. We reduce max weight to 24 and use hints from both of the previous jobs.

Squeezing the max weight III

Length 157, Level 42, Max wt 24, Given 246, Exp length 384

Now keep max weight at 24, and use the hints just from the previous job.

Length 144, Level 39, Max wt 24, Given 260, Exp length 333

Keep max weight at 24, repeat

Length 153, Level 43, Max wt 24, Given 253

This is going in the wrong direction, so we discard this one.

We try squeezing max weight down to 23.

Length 153, Level 44, Max wt 23, Given 144, Exp length 358

Keep max weight at 23 and repeat:

Length 138, Level 41, Max wt 23, Given 289, Exp length 316

This proof is stable!

If we reduce max weight to 22, then we get no proof after 2000 givens.

Pushing proofs off the hints I

Now that we have done our best with max weight, let's see what happens if we force Prover9 to ignore the hints from time to time.

```
list(given_selection).
  part(H, low, weight, hint) = 10.
  part(A, low, age, all) = 1.
  part(T, low, weight, true) = 1.
  part(F, low, weight, false) = 1.
end_of_list.
```

Length 127, Level 32, Max wt 23, Given 316, Exp length 286.

Pushing proofs off the hints II

Now restore defaults for given selection and see if we get anywhere “naturally”.

Length 116, Level 32, Max wt 23, Given 316, Exp length 260

This is stable.

This is the best I was able to do with this problem. I tried again the trick of reducing hints to low priority, but each attempt either gave no proof (after 2000 givens) or a more complex proof than this last one.

Other tricks

None of these helped in this case, but they sometimes do.

- Switch from selecting hints by weight to selecting by age.
- Add “obvious” steps in the proof to the input.
- Mix up the function order.
- Drop the semantic guidance. Then put it back.