Efficient encodings of first-order Horn problems in equational logic

Koen Claessen
Nick Smallbone

Equational theorem provers are awesome!

High performance!

Readable proofs!

But there's a problem…

Reasoning about algebra

$$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z \qquad x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$$

$$x \sqcup y = y \sqcup x \qquad x \sqcap y = y \sqcap x$$

$$x \sqcup x = x \qquad x \sqcap x = x$$

$$x \sqcup (x \sqcap y) = x \qquad x \sqcap (x \sqcup y) = x$$

$$x \sqcup \bar{x} = \top \qquad x \sqcap \bar{x} = \bot$$

$$x \sqcap (y \sqcup (x \sqcap z)) = (x \sqcap y) \sqcup (x \sqcap (y \sqcup (z \sqcap (x \sqcup (y \sqcap z)))))$$

$$x \sqcap y = \bot \wedge x \sqcup y = \top \rightarrow \bar{x} = y$$

$$a \sqcup b = b$$

$$\bar{a} \sqcap \bar{b} \neq \bar{b}$$

Reasoning about algebra

$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$

$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$

$x \sqcup y = y \sqcup x$

$x \sqcap$

$x \sqcup x = x$

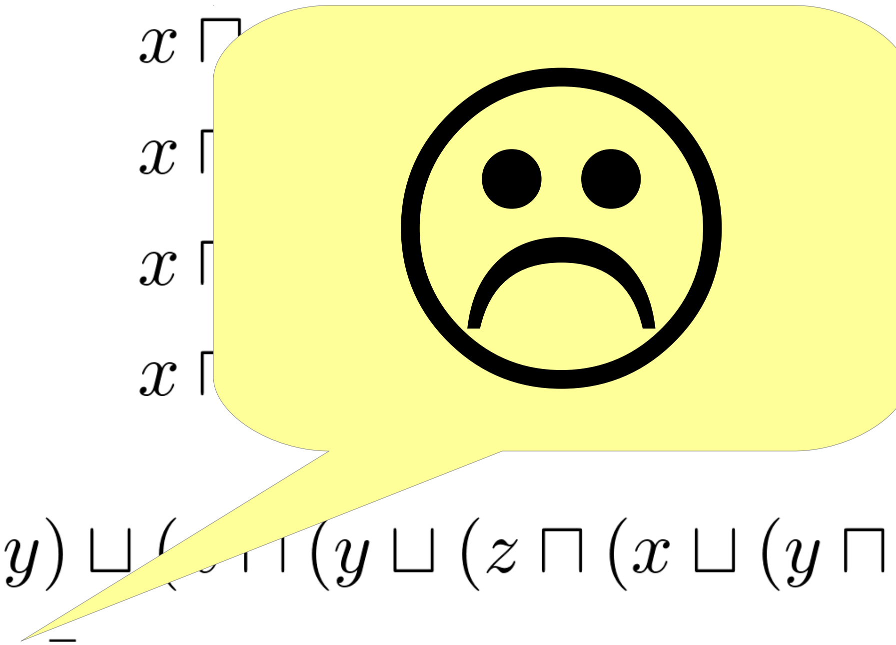
$x \sqcap$

$x \sqcup (x \sqcap y) = x$

$x \sqcap$

$x \sqcup \bar{x} = \top$

$x \sqcap$



$x \sqcap (y \sqcup (x \sqcap z)) = (x \sqcap y) \sqcup ($ $(y \sqcup (z \sqcap (x \sqcup (y \sqcap z)))))$

$x \sqcap y = \bot \wedge x \sqcup y = \top \rightarrow \bar{x} = y$

$a \sqcup b = b$

$\bar{a} \sqcap \bar{b} \neq \bar{b}$

Reasoning about functional programs

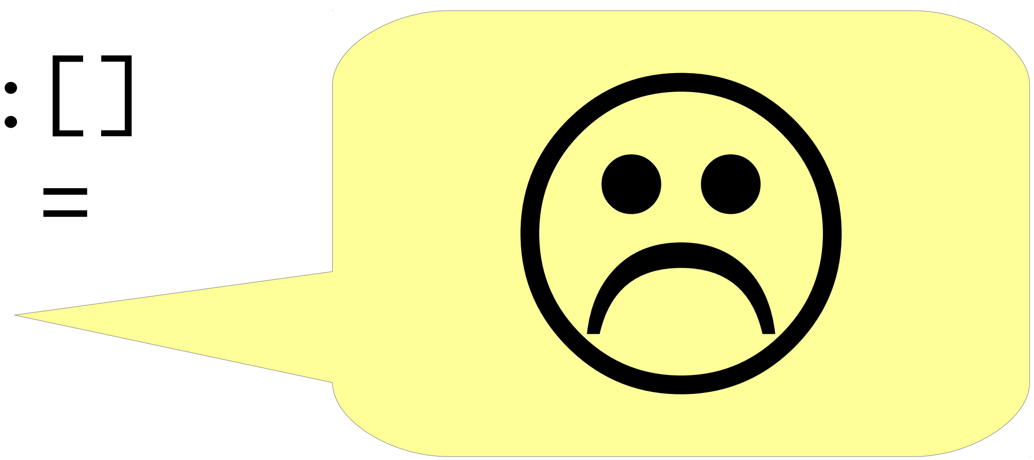
```
sort [] = []
sort (x:xs) = insert x (sort xs)

insert x [] = x:[]
insert x (y:ys) =
  if x < y then
    x:y:ys
  else
    y:(insert x ys)
```

Reasoning about functional programs

```
sort [] = []
sort (x:xs) = insert x (sort xs)

insert x [] = x:[]
insert x (y:ys) =
  if x < y then
    x:y:ys
  else
    y:(insert x ys)
```



Our paper

What? Teach a unit equality prover how to reason about Horn clauses

How? By encoding Horn clauses as equations (leaving existing equations alone)

Why? To get a prover that specialises in *mostly-equational* reasoning

The plan

Horn clause: at most one positive literal

- e.g. $f(x) = x \land p(x) \rightarrow g(x) = a$
- or $p(x) \land q(x) \rightarrow \text{false}$

Stage 1: how to encode clauses of the form:

$$a = b \rightarrow c = d$$

Stage 2: how to encode clauses of the form:

$$a_1 = b_1 \land \ldots a_n = b_n \rightarrow c = d$$

Stage 3: how to encode predicates

Skipped: how to encode goal clauses

Stage 1:

How to encode a binary clause

$(a = b \rightarrow c = d)$

as an equation

The idea

We are going to axiomatise a function ifeq:

$$\mathsf{ifeq}(x, y, z, w) = \text{if } x = y \text{ then } z \text{ else } w$$

Then we can encode $a = b \rightarrow c = d$ as:

$$\mathsf{ifeq}(a, b, c, d) = d$$

If $a = b$ then $\mathsf{ifeq}(a, b, c, d) = c = d$;
if $a \neq b$ then $\mathsf{ifeq}(a, b, c, d) = d$.

But how to axiomatise ifeq equationally?

Axioms for ifeq

☺ Axiom 1:

$$\mathrm{ifeq}(x, x, y, z) = y$$

☹ Axiom 2:

$$x \neq y \rightarrow \mathrm{ifeq}(x, y, z, w) = w$$

For Horn formulas, we only need the first axiom!

How to discharge a conditional

Given $a = b \rightarrow c = d$:

$$\mathrm{ifeq}(a, b, c, d) = d$$

$$\mathrm{ifeq}(x, x, y, z) = y$$

If $a = b$ then

$$\mathrm{ifeq}(a, b, c, d) = d$$
$$||$$
$$\mathrm{ifeq}(a, a, c, d) = c$$

so $c = d$.

This deduction rule (positive unit resolution) is enough for Horn reasoning

Stage 2:

How to encode an *n*-ary clause
(e.g. $a{=}b \wedge c = d \rightarrow e{=}f$)
as an equation

Clauses with many negative literals

Option #1: repeatedly apply the encoding

a = b & **c = d → e = f**

becomes

a = b → **ifeq(c, d, e, f) = f**

becomes

ifeq(a, b, ifeq(c, d, e, f), f) = f

Clauses with many negative literals

Option #2: tupling

$a = b \ \& \ c = d \rightarrow e = f$

becomes

$\mathrm{pair}(a, c) = \mathrm{pair}(b, d) \rightarrow e = f$

becomes

$\mathrm{ifeq}(\mathrm{pair}(a, c), \mathrm{pair}(b, d), e, f) = f$

where pair must be fresh.

(this is not sound for non-Horn problems!)

- Example: LAT224-1

Stage 3:

How to encode predicates
using equations

Encoding predicates

Idea: add a constant true and encode p(t) as p(t) = true

Encoding predicates

Idea: add a constant true and encode p(t) as p(t) = true

Unsound!

Encoding predicates

Idea: add a constant true and encode p(t) as p(t) = true

Satisfiable: $(\forall x, y.\ x = y) \land \neg p$

Unsatisfiable: $(\forall x, y.\ x = y) \land p \neq \text{true}$

Solution #1: use types

Solution #2: ...

Encoding predicates

Satisfiable: $(\forall x, y.\ x = y) \land \neg p$

Unsatisfiable: $(\forall x, y.\ x = y) \land p \neq \mathrm{true}$

Encoding predicates is *only* unsound because you can go from $\forall x, y.\ x = y$ to $p = \mathrm{true}$

- this is only a problem if $\forall x, y.\ x = y$ is provable but the formula is satisfiable
- i.e., when the problem has a model of domain size 1

So, *first* check if there is a model of size 1

- Easy-peasy: predicates are constant-valued, and all equality literals are true, so this is a HORNSAT problem

If there is a model of size 1, the problem is satisfiable

Otherwise, encoding predicates is sound

- If $\forall x, y.\ x = y$ is provable, the problem is unsatisfiable

An alternative encoding

The ifeq-encoding is suboptimal

$$\mathrm{ifeq}(a, b, c, d) = d$$

The prover only needs to make a and b equal

But it will also needlessly reason about c and d

Encoding #2

$a = b \rightarrow c = d$ becomes:

$$\mathrm{fresh}(a, x_1, \ldots, x_n) = c$$

$$\mathrm{fresh}(b, \underbrace{x_1, \ldots, x_n}) = d$$

A fresh function symbol

All free variables of a, b, c and d (*substitution)*

Encoding #2: $a = b \rightarrow c = d$

Given $a = b \rightarrow c = d$:

$$\mathrm{fresh}(a, x_1, \ldots, x_n) = c$$

$$\mathrm{fresh}(b, x_1, \ldots, x_n) = d$$

If $a = b$ then

$$\begin{array}{c}\mathrm{fresh}(a, x_1, \ldots, x_n) = c \\ \parallel \\ \mathrm{fresh}(b, x_1, \ldots, x_n) = d\end{array}$$

so $c = d$.

Encoding #2

$$\mathrm{fresh}(a, x_1, \ldots, x_n) = c$$
$$\mathrm{fresh}(b, x_1, \ldots, x_n) = d$$

If these equations are oriented left-to-right then the prover can only:

- paramodulate into a
- paramodulate into b
- deduce c=d once a and b are shown equal

This is about what we'd get with a dedicated prover!

Downside: now have *two* clauses to reason about

Results

Prover	Solved	Rating 1 solved
E (not encoded)	**1972**	0
E (encoded)	1710	7
Twee	1683	**29**
Waldmeister	1378	15
SPASS (not encoded)	1370	0

37 problems of rating 1 solved in total, all are heavily equational
- Out of **120** rating 1 Horn problems total

Conclusion: the equational provers are respectable (but not great) at general-purpose reasoning – but very good at equational reasoning!

Possible heuristics

… ifeq(s, t, u, v) …

1. Prioritise inferences that make the first two arguments equal
   - Solves more rating 1 problems but fewer problems overall
2. Don't do any inferences on the third and fourth argument
   - Not implemented, but ought to help

More?

Full first-order logic?

$a = b \vee c = d$

is the same as

$\mathrm{ifeq}(a, b, c, d) = c$

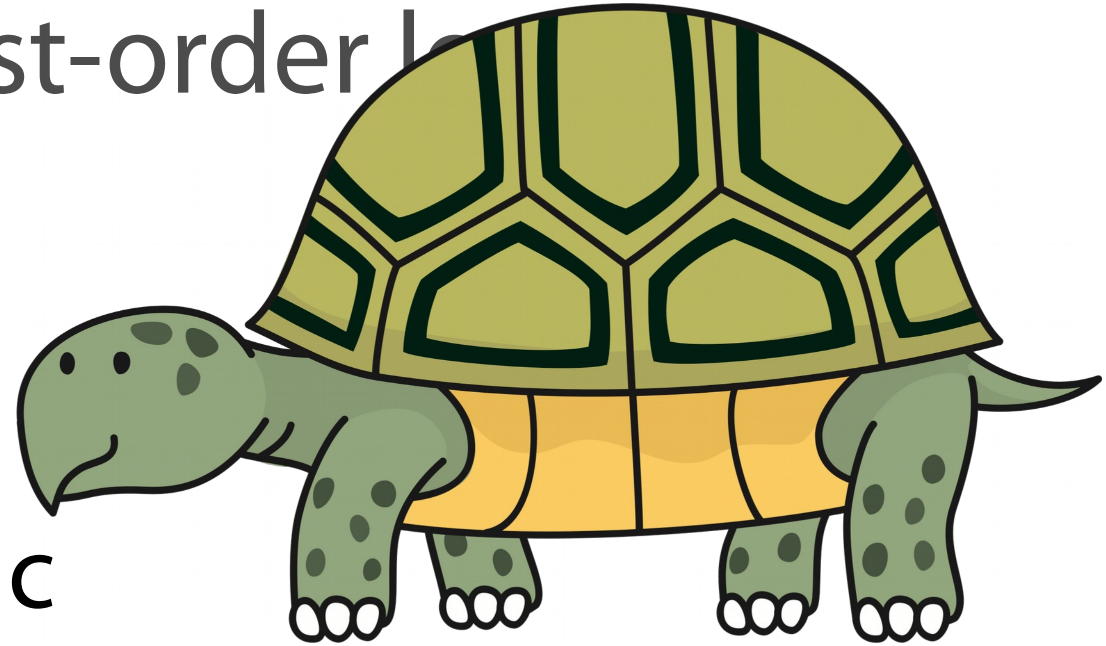
but *only* if we can axiomatise

$x \neq y \to \mathrm{ifeq}(x, y, z, w) = w$!

It's known how to do this… with about 10 different axioms plus congruence axioms for each function symbol

Full first-order l

Help!
I need some heuristics



ifeq(a, b, c, d) = c

but *only* if we can axiomatise

$x \neq y \rightarrow \text{ifeq}(x, y, z, w) = w$!

It's known how to do this… with about 10 different axioms plus congruence axioms for each function symbol

Conclusions

Goal: a cheap way to turn an equational prover into an "equational-plus" prover

- Encodings work well for mostly-equational problems
- Teaching the prover extra heuristics seems to be a good idea
- Can we adjust the prover's strategy a bit more radically? (Ignoring certain inferences, always choosing certain inferences, ...)
- Full first-order logic – can it be done?
- Decoding proofs?

In the paper: four encodings, how to handle conjectures, proofs of correctness

You can try it out using Twee (on TPTP)

- ...and also watch Twee lose at CASC!