Practical Mechanical Theorem Proving

Tom Ridge

December 4, 2003

School of informatics

Overview

- Mathematicians are (quite) interested in computers
  - but they are not (yet) interested in mechanised theorem proving
- Intel, AMD and others are interested in mechanised theorem proving
  - because there are (economic) drivers compelling them to "get it right"
- The proofs in these areas are NOT typical mathematical proofs
  - they are long and boring and so ideally suited for mechanical verification

Overview

On the other hand . . . even (human produced) mathematical proofs are becoming extremely long, and referees are struggling.

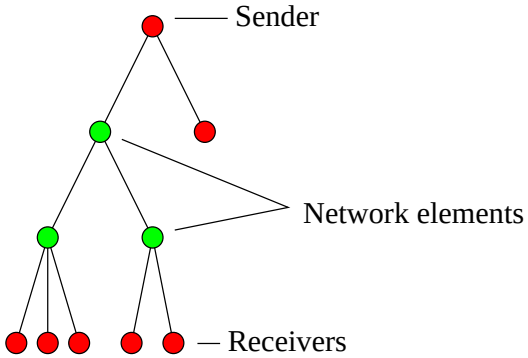
See extras section detailing flyspeck project. Referees trying to verify the result came up against a very big proof:

> They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem.

The PGM Protocol

- IETF RFC 3208, draft internet standard
- A Sender, at the root of a *tree* network, sends messages
- *Receivers*, at the leaves of the tree, receive messages via. intermediate *network nodes*
- Would like to know, for instance, that

  for all receivers r, if the sender s multicasts a message m (identified by some sequence number), then eventually r receives m, or r realises that m has been lost.



Modelling The Protocol I: State Machine

- Take a model from a current research paper [EM03]
- Model essentially that of a state machine
- Mechanise proofs from paper (proofs very complicated)
- Model very concrete, so results not very general
- Abstract to get at important properties/ make proofs more general

See extras for a description of the agent for the Sender. In particular, note that the description is mathematical/ logical, so we can reason about it, and computational, so we can program it.

Modelling The Protocol II: Events

- Actions/ events occur at a particular node n in a tree (constraint: n must be a node in the tree)
- A message i is taken from an adjacent channel c (constraint: c must be a channel connecting n to another node)
- A message $o1$ possibly sent upwards
- A message $o2$ possibly sent downwards through a subset of downwards channels C (constraint: C must be a subset of n's downward channels)

See extras for an extract from the proof. Frightening!

Modelling The Protocol in Isabelle/HOL

- Follow the model in the paper as closely as possible
- but paper isn't a formal object, so may not always be possible
- Also need to develop libraries to deal with e.g. trees
- but very labour intensive. . . so try and reuse theories already developed
- Very steep learning curve
- Eventual proof took considerable effort and perseverance, but experience invaluable

Library example (mathematical): Trees

- Lots of ways to model trees: various datatypes, various set models
- It pays to *stick very closely to mathematical practice*
- Most mathematical definitions of trees start from the notion of a *graph* as a (loopfree) set of vertices and edges
- A path is a list of distinct nodes (which are adjacent in the graph)
- A tree is an *acyclic*, *connected* graph
- Good choice of representation is *the overriding contributor* to whether mechanisation attempt is successful

Trees in Isabelle

lemma (**in** *trails-etc*) *is-path-def-2*: $is\text{-}path\ p = (p \neq [] \wedge distinct\ p)$

lemma *is-cycle-def-2*: $is\text{-}cycle\ p = (hd\ p = last\ p \wedge 4 \leq length\ p \wedge distinct\ (tl\ p))$

record $'a\ pre\text{-}graph = \quad Verts :: 'a\ set \quad Edges :: 'a\ edge\ set$

constdefs $is\text{-}graph :: ('a,'b)\ pre\text{-}graph\text{-}scheme \Rightarrow bool$
$is\text{-}graph\ g \equiv Edges\ g \subseteq Pow\ (Verts\ g) \wedge (\forall\ x.\ \{x,x\} \notin (Edges\ g))$

constdefs $is\text{-}tree :: ('a,'b)\ pre\text{-}tree\text{-}scheme \Rightarrow bool$
$is\text{-}tree\ t \equiv is\text{-}graph\ t \wedge is\text{-}connected\ t \wedge is\text{-}acyclic\ t$

constdefs $is\text{-}rooted\text{-}tree :: ('a,'b)\ pre\text{-}rooted\text{-}tree\text{-}scheme \Rightarrow bool$
$is\text{-}rooted\text{-}tree\ t \equiv is\text{-}tree\ t \wedge root\ t \in Verts\ t$

constdefs $ch\text{-}up :: ('n,'b)\ pre\text{-}rooted\text{-}tree\text{-}scheme \Rightarrow 'n \Rightarrow 'n\ edge$
$ch\text{-}up\ t\ v \equiv \{v,parent\ t\ v\}$

Representation

- There are literally 1000's of choices when choosing how to represent things
- For example, all functions in HOL are total (= defined everywhere)
- . . . so we have to decide how to represent functions that are undefined for some points in their domain
- e.g. what is hd []? what is tl []?
- Sometimes there is a mathematically natural element we can choose to represent undefined (tl [] = []). Or this may be suggested by desire to abstract and use symbols. For instance, we *define* $x^0 = 1$.

- Alternatively we define it to be some element, about which we know nothing (hd [] = arbitrary)
- Other alternatives: lift every domain to have an element representing "undefined", use a logic with undefinedness built in (IMPS), represent definednes in the type etc.
- Maybe tl [] = [] not natural, but allows some simps to work without making case distinctions.

lemma *length-tl* [*simp*]: $length\ (tl\ xs) = length\ xs - 1$ **by** $(cases\ xs)\ auto$

- In our case, do we want [] to be a valid path? Maybe we just consider it as a valid path so that our simps aren't always conditional on well formedness

Example: $Ch\uparrow$, and guarded definitions

- parent not defined for Sender, so $Ch\uparrow$ not defined for Sender
- Edges are sets of vertices: an edge from x to y is $\{x, y\}$
- Natural element "empty set" to identify undefined channel, so could define $Ch\uparrow\ Sender = \{\}$ (but note definition is ugly)
- We know we will only use $Ch\uparrow$ in situations where it has a meaning, so leave $Ch\uparrow\ Sender$ arbitrary
- Must enforce usage of $Ch\uparrow$ so doesn't apply to Sender!

lemma (**in** *loc-Is-event*) *Is-event-Ev*[*simp*]: *Is-event* (*Ev n i c o1 o2 C*) =
(*let Sender* = *R in*
$n \in V$
$\wedge$ *Is-action* (*i*,*o1*,*o2*)
$\wedge$ ($n = Sender \longrightarrow c \in Ch\downarrow\ n$)
$\wedge$ ($n \neq Sender \longrightarrow c \in \{Ch\uparrow\ n\} \cup Ch\downarrow\ n$)
$\wedge\ C \subseteq Ch\downarrow\ n$
$\wedge$ (($i \in M\uparrow \wedge c \in Ch\downarrow\ n$)
$\vee$ ($i \in M\downarrow \wedge n \neq Sender \wedge c = Ch\uparrow\ n$)
$\vee$ ($i = \perp$)))

Note that each use of $Ch\uparrow\ n$ is guarded by a condition asserting $n \neq Sender$.

Example: Empty Event

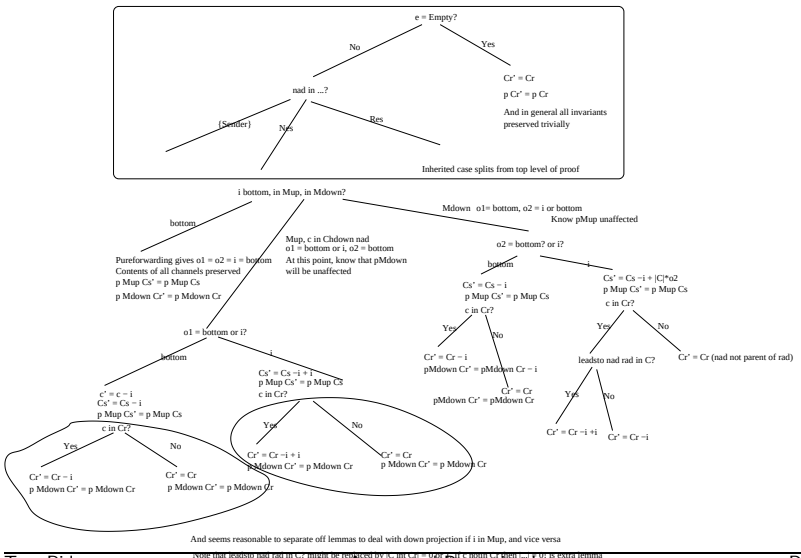
- Model an event as a tuple $(n, i, c, o1, o2, C)$
- But also need an empty event "distinct from all the others" (for each node?)
- Unlike $Ch\uparrow\ Sender$, we DO need to know when an event is an empty event: it is properly part of the model
- So maybe choose $(n, \perp, Ch\uparrow\ n, \perp, \perp,)$ to represent an empty event, and have an associated predicate is_empty_event
- Our proofs look fine. . .
- . . . but $Ch\uparrow$ not defined for Sender

- Does this matter? It means that we cannot tell whether $(Sender, \perp, c, \perp, \perp,)$ (c some down channel for the Sender) is not the empty event (because to do so we would have to prove that $c \neq Ch\uparrow\ Sender$).
- In some sense, we could construct our proofs, and they would look exactly as we want them to. The problem comes when we try and apply them to a real example.
- Settle on having an empty event as a separate part of a datatype

datatype (*'name*, *'msg*) *event* = *Empty*
| *Ev 'name 'msg 'name channel 'msg 'msg 'name channel set*

Planning The Mechanisation

- Make sure you understand the proof BEFORE you mechanise! Unfortunately TP systems are not at the stage where they can actually HELP you with the proof
- Plan the proof on paper, down to the last detail
- Try and isolate all the intermediate lemmas the proof depends on
- Try and make your job easier by proving only what you have to



Natural deduction and Readability

- Natural deduction is the standard way to present proofs
- Also seems to deserve its name: in some sense it is "natural"
- Isar implements natural deduction style human readable proofs
- Scoped, hierachical structure- (the only?) general method for organisation
- One of the aims of my current work was to investigate to what extent the presentation of mathematical proofs can be rendered in Isar

See extras "Diagonalisation" for an example of natural deduction/ readable proofs.

Automation

- Simplification well understood, proof search quite well understood
- Combination not so well understood
- But automation can be highly effective (this surprised me)
- And if you want to preserve your sanity, its worth understanding

See extras for an example of the diagonalisation proof conducted with the help of automated tools. Which proof is preferable?

HOL- expert level stuff

Finally, for the real afficionados, it's worth understanding the ins and outs of the logic you are using, since this can lead to incredibly slick proofs. For example, mathematicians frequently refer to inductively defined sets as "the least set containing x and closed under y" (x a constant, y some operation). How can we use this definition to derive an induction principle?

See Induct.thy in extras for a truly beautiful example, where defining a set is the same thing as defining an induction principle.

Summary

- Theorem proving like programming to some extent, but much more arduous
- Thousands of opportunites to go wrong
- Requires intelligence at almost every step of the way
  - and sometimes almost superhuman levels of perseverance
- But rewards are substantial, and future is bright
- If you are interested in truth, proof is a good place to start!

See [Rid] for this document and related theory files.

References

[EM03] Javier Esparza and Monika Maidl. Simple representative instantiations for multicast protocols. In *TACAS 2003*, pages 128–143. Springer-Verlag LNCS 2619, 2003. http://www.dcs.ed.ac.uk/home/monika/maidl-pgm-reduction.ps.

[Rid] Tom Ridge. Informatics homepage. http://homepages.inf.ed.ac.uk/s0128214/.