Interfacing ITP to other tools and the real world

(a few bullet points of possibly incoherent, but discussion-provoking, crazy ideas)

Peter Sewell

University of Cambridge

http://www.cl.cam.ac.uk/~pes20/

WITP, 25 August 2009

Interfacing ITP to other tools and the real world

(a few bullet points of possibly incoherent, but discussion-provoking, crazy ideas)

Peter Sewell

University of Cambridge

http://www.cl.cam.ac.uk/~pes20/

WITP, 25 August 2009

Interfacing ITP to other tools and **the real world**

(a few bullet points of possibly incoherent,
but discussion-provoking, crazy ideas)

Peter Sewell

University of Cambridge

http://www.cl.cam.ac.uk/~pes20/

WITP, 25 August 2009

Some ‘Real-World’ Applications of ITP

- Network protocols (TCP, SWIFT) (Norrish, Ridge,...)
- Programming language semantics
  - POPLmark (Pierce, Weirich, Zdancewic,...)
  - The Ott tool, compiling PL definitions (Zappa Nardelli,...)
  - Java Module Systems (Strniša)
  - An OCaml fragment (OCaml_light) (Owens)
  - A verified executable distributed queue (Ridge)
- Multiprocessor and C++ concurrency semantics (x86, PPC, ARM) (Sarkar, Owens, Ridge, Zappa Nardelli, Myreen, Fox, Alglave, Maranget, Batty,...)

Some ‘Real-World’ Applications of ITP

- Network protocols (TCP, SWIFT) in HOL
- Programming language semantics
  - POPLmark
  - The Ott tool, compiling PL definitions to LaTeX, Coq, HOL, and Isabelle/HOL
  - Java Module Systems in Ott and Isabelle/HOL
  - An OCaml fragment (OCaml_light) in Ott and HOL
  - A verified executable distributed queue in HOL
- Multiprocessor and C++ concurrency semantics (x86, PPC, ARM) in HOL and Coq

Crazy Idea #1

Crazy Idea #1

ITP tools are great!

Looking at those Applications

- Coming up with the definitions is a major part of the work.
  - They’re big (1000s or 10 000s of lines) and complicated.
  - They’re of key (and relatively stable) CS abstractions

They should be *reusable* artefacts

Looking at those Applications

- Mechanised proof is not always the point.

  Sometimes:

  - no proof (typechecked typeset maths)
  - mechanised symbolic evaluation...
  - ...and code generation (for testing and prototyping)
  - hand proof
  - mixed hand and mechanised proof
  - full mechanised proof

Looking at those Applications

- They’re logically undemanding:
  - no need for dependent types or type classes
  - we typically don’t *care* whether we’re classical or constructive
  - there’s not much object-language variable binding
    (not true for fancier PLs, though)
- we do make heavy use of “PL” types: inductive types and records, and functions and relations over them

Looking at those Applications

- The ITP tool is just one piece of a complex ‘workflow’:
  - production typesetting
  - testing infrastructure
  - target for code generation (Ott)
  - target for auto-embedding of source language terms

Looking at those Applications

- We have to use *multiple* ITP tools:
  - To fit in with local expertise (in multiple sites!)
  - To make resulting models widely available

Crazy Idea #2

Crazy Idea #2 (mindset)

Your ITP tool is not at the centre of the (user's) world

Crazy Idea #3 (ITP?)

They may not be using it interactively

Crazy Idea #4 (ITP?)

It’s not all about proofs. Definitions are (sometimes) more central!

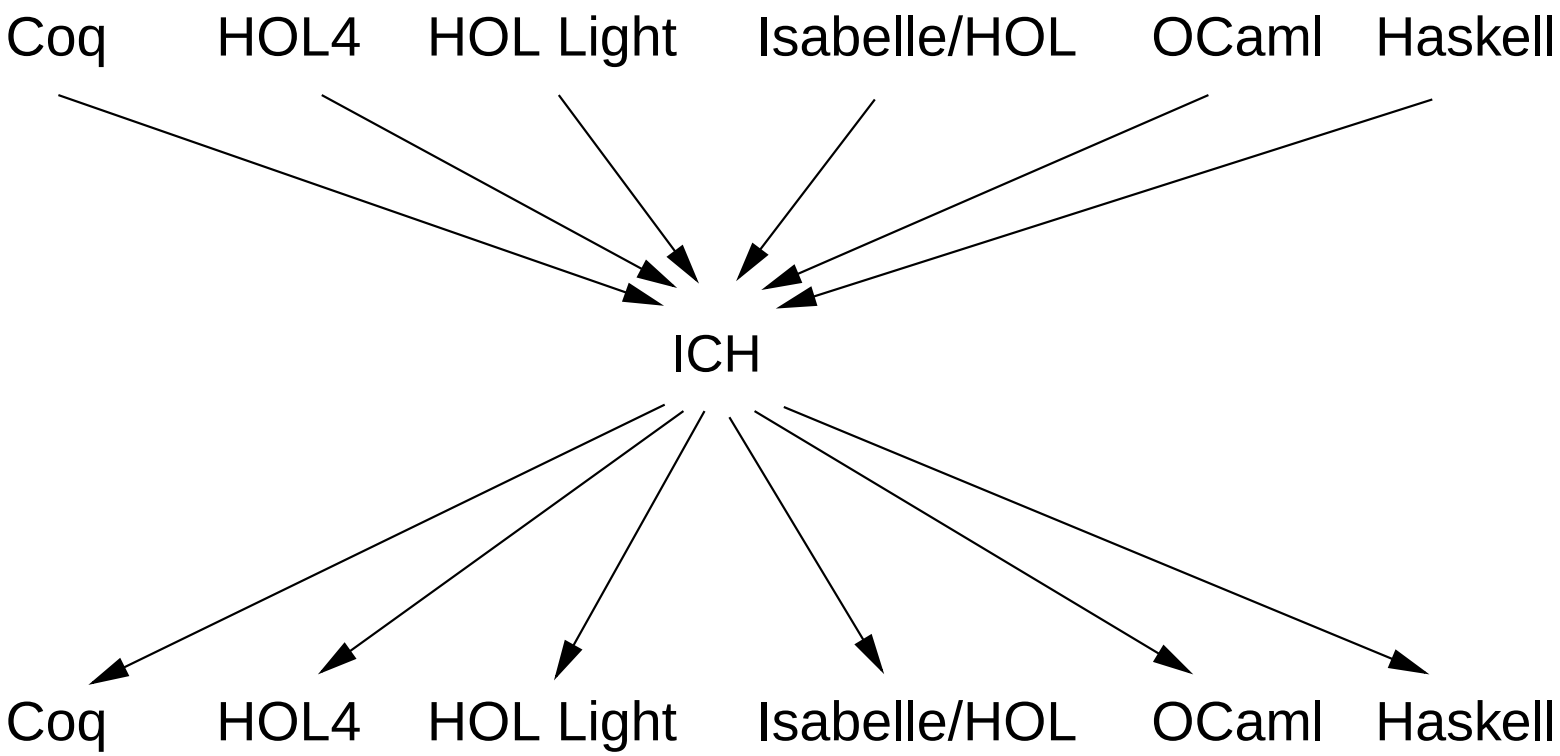
Crazy Idea #5: Lightweight Translation of Definitions

Urgently needed: lightweight support for *translating* these big definitions between ITP tools, to make them reusable.

Coq HOL4 HOL Light Isabelle/HOL OCaml Haskell

Crazy Idea #5: Lightweight Translation of Definitions

Urgently needed: lightweight support for *translating* these big definitions between ITP tools, to make them reusable.



In general, obviously impossible. But in many cases, it should be (a) easy, and (b) staggeringly useful. Sketch plan:

1. translate source files (idiomatic, readable).
   NOT proof scripts or proof terms.
2. define that ICH intermediate language
   - roughly the intersection of Coq/HOL/Isabelle-HOL
   - but including source-level definitions of types, functions, relations, etc., not the kernel logics
   - specify type system and *abstract* syntax
   - sort out libraries for sets, lists,...
3. take prover source files and export ICH code
   (in the provers — this needs you!)
4. take ICH code and output prover source files (easy)

Crazy Idea #5: Lightweight Translation of Definitions

Then also use that intermediate language as a target for tools like Ott, LNgen, and object-language embeddings.

Crazy Idea #5: Lightweight Translation of Definitions

Then also use that intermediate language as a target for tools like Ott, LNgen, and object-language embeddings.

and watch as, somewhat before the *next* millennium, CS becomes based on reusable de-facto-standard mechanised artifacts...