ITP Uses and Challenges at Rockwell Collins

August 24 2009

Konrad Slind
Rockwell Collins Advanced Technology Center
Cedar Rapids, IA

Building trust every day

Rockwell Collins' core business is based on delivery of *High Assurance* Systems

- Commercial/Military Avionics Systems
- Flight Control Systems
- Heads Up Displays
- Navigation & Landing Systems
- Defense Communications

"Working together creating the most trusted source of communication and aviation electronic solutions"

HOL and ACL2

- Interactive theorem provers with a long pedigree
- Separate user groups, culture, and focus
- ACL2 : recursive mathematics in seemingly unquantified FOL
- HOL : higher order logic with simple types
- Example: divisibility.

```
ACL2:  divides a b = x <> 0 /\ integerp (y / x)
       least-divisor k n =
          if integerp(n) /\ integerp(k) /\ 1 < k <= n
            then if divides k n then k else least-divisor (k+1) n
            else nil
      prime(p) = integerp(p) /\ (least-divisor 2 p = p)
```

```
HOL:  divides a b = ?d. b = a * d
    prime p = (p <> 1) /\ !n. divides n p ➔ (n=1) \/ (n=p)
```

Impressions of ACL2

- Declarative proof is nice! Can start getting results right away.
- Learning curve has few handholds
- Implicit context is un-nerving (every previously proved fact is by default in the implicit context)
- Impressive online documentation
- I keep forgetting to set rule classes on proved theorems, which causes later proofs to fail
- Reading failed proof transcripts is depressing ("the method")
- Nostalgic for types.
- However, defining functions to work over the whole ACL2 universe is engaging once you understand a few basics.
- Monotonicity fails

Monotonicity

- At the level of deduction:
  - If Gamma |- A then Gamma, B |- A
- At the level of theory development
  - If Context |- A then Context,B |- A
- Having more info in context can derail existing proofs
- When monotonicity fails, proof developments tend to become "append only"
- Large-scale formalization steps, e.g. merging libraries, become more fragile
- BUT
- Implicit context v. helpful in controlling complexity of interaction

Computation

- Of the systems I've used, ACL2 treats the idea of computation most extensively.
- Evidence : executable counterparts, guards, mbe, stobj
- Seamless passage of functions and results back and forth between OL and ML.
- Only an implicit notion of computable function
- Logical functions do not have an operational semantics visible inside the logic or (alternatively) a visible EVAL
- The logic is a theory of s-expressions and those are identified (fully?) with the s-expressions of the ML.
- What would something like this look like for other systems?
- Possible starting point: an SML that had HOL types and terms as primitive?

Cultures

- Each prover has a high barrier to entry
- Logic is the least of it!
- HOL concepts: rule (primitive and derived), tactic, conversion, theory, library, plus vagaries of host ML.
- Isabelle concepts: rule (primitive and derived), h.o. unification, type class, locale, ISAR language
- ACL2 concepts: book, hints, rule classes, guards, mbe, stobj
- Behaviour of reasoners with hidden state (rewriters especially)
- BUT
- Ancient systems always provide a way to emulate behaviour (decision procedures as derived rules, rule-classes nil)
- Turing tarpit: computation in the ML can bridge gaps
- High degree of viscosity: people get invested (compare with SAT or SMT)

Theory structuring mechanisms

- HOL: theory segments, DAG of
- ACL2: books
- Meeting ground between software engineering and logic
- Issue: library development concurrent with development of theories using library.
- Issue: dependency maintenance. With separately compiled theories comes Makefiles. Tends to be a horror show (“do I have GNU make on this machine, or what?” etc). We wrote our own. Does everybody write their own?
- Issue: quarreling theories. Theories A and B overlap, but each offers significant functionality that the other doesn’t (e.g. proof automation or difficult theorems). But it is difficult to use both at the same time. Usually can be worked-around, though painful.

My ITP wish list

- If I know a proof in detail, I want to be able to get the proof system to do **that** proof. Without having to tinker extensively or drop down to an overly low level of interaction.
- If my conditional rewriter can't prove a condition and I really do want that rewrite to complete, then I should be able to force the rewrite and get the condition appearing as an extra proof obligation. (Peter Homeier's `dependent rewriting').
- In the middle of a proof I want to be able to add new facts, by asserting them on the spot and having the system prove them or by referring to previously proved facts.
- System should tell me at least something that is missing from failed proof attempt.
- What we are doing almost all the time is dealing with failure and trying to garner information that will show the cause of failure.

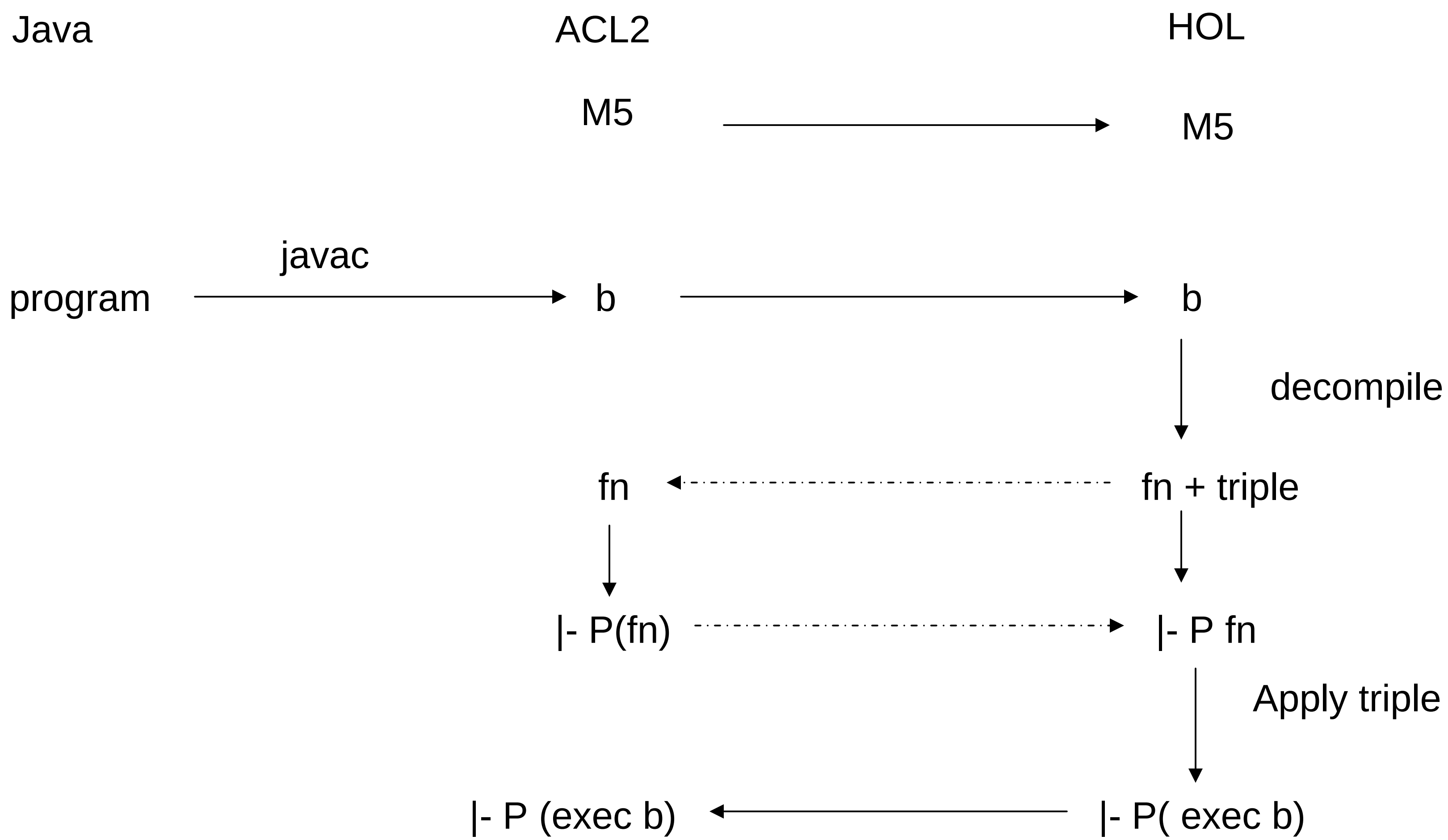
HOL—ACL2 interaction

- See the work by Matt Kaufmann and Mike Gordon
- The artifact exists. How can it be exploited?
- Two discernible starting points:
  - There's a difference in expressive power, so use HOL to formalize abstract notions. Use ACL2 in its sweet spot.
  - There's a less intrinsic difference, e.g. one system has a large formal model that the other lacks; or provides proof support that the other lacks; or a user is simply unwilling to learn a new system.
- Typically want to either make a case that the task can't be accomplished any other way, or that it is interesting that the task be broken across two proof systems.
- Compare with old QED proposal
- Compare with current mechanisms for sharing theories between proof systems.

Possible Application: bytecode proofs

- ACL2 has (thanks to J) a detailed JVM model
- HOL-4 has (thanks to Magnus) a decompiler
  - Decompile : assembly -> recursive fn + triple
  - Triple asserts that running asm on input equals fn on input
- Observation: direct verification of bytecode is too time-consuming and detailed
- Idea: use decompiler on bytecodes to see if reasoning about rec. fns can be more productive

Bytecode proof flow



The End

Thank you!