Towards a Combination of Heterogeneous Deductive Tools for System Verification

Pacal Fontaine  Kamal Gupta  Jean-Yves Marion
Stephan Merz  Leonor Prensa Nieto  Alwen Tiu

LORIA, Nancy, France
October 14 - 15, 2005
Motivations

- Tradeoff between the expressivity of specification languages and support for automation:
  - Proof assistants (Isabelle/HOL, Coq, ...) are very expressive, provide strong guarantee of consistency (with explicit proof objects), but lack automation.
  - Automated proof tools (model checkers, SAT solvers, etc.) are efficient but have limited expressivity, and prone to bugs.
- Is it possible to combine the two approaches to verification? Maybe, for a specific domain of applications, e.g., verification of distributed algorithms.
Outline of a platform for tools integration

- Basically, a meta logic acts as a *proof manager*.
  (Isabelle/HOL)

- The *specification language* can be the meta logic itself
  (HOL) or other specific logics or computation models (e.g.,
  TLA, I/O automata, etc.) encoded inside it.

- Some parts of the proofs (in Isabelle/HOL) can be given to
  external automated provers to be solved automatically.
Tools integration

- Ideally, external tools produce proof traces. Proofs are then reconstructed inside Isabelle/HOL.
- We focus on verification of distributed algorithms. Case study: fault-tolerant clock synchronization.
Tools integration

- Ideally, external tools produce proof traces. Proofs are then reconstructed inside Isabelle/HOL.
- We focus on verification of distributed algorithms. Case study: fault-tolerant clock synchronization.
Case study: fault-tolerant clock synchronization

- We consider verifying a generalized fault-tolerant clock synchronization algorithm by Schneider [Schneider’87, Shankar’92].
- The proof involves real-number arithmetic and first-order reasoning. Hence suitable for testing combination with automatic arithmetic solvers.
- The proof is first formalized in Isabelle/HOL (around 1400 lines using Isar proof script), and some crucial lemmas are given to automated tools (ICS and CVC-lite).
Example: proving bounded clock drift (1)

Physical clocks are constantly drifting away from the “real” time.

\[\text{rbound1 } C \triangleq \forall p, s, t. \text{ correct } p \ t \land s \leq t \]
\[\implies C \ p \ t - C \ p \ s \leq (t - s) \times (1 + \rho)\]

\[\text{rbound2 } C \triangleq \forall p, s, t. \text{ correct } p \ t \land s \leq t \]
\[\implies (t - s) \times (1 - \rho) \leq C \ p \ t - C \ p \ s\]

\(\rho\): drift rate.
\(C \ p \ t\): clock reading of process \(p\) at real time \(t\).
\(\text{correct } p \ t\): process \(p\) non-faulty at real time \(t\).
Example: proving bounded clock drift (2)

**Lemma** bounded-drift:

- **Assumes** \( s \leq t \) and **correct** \( p \ t \) and **correct** \( q \ t \)
- and **rbound1** \( C \) and **rbound2** \( C \)
- and **rbound1** \( D \) and **rbound2** \( D \)
- **Shows**

\[
| C \ p \ t - D \ q \ t | \leq | C \ p \ s - D \ q \ s | + 2 \times \rho \times (t - s)
\]

- Proved by CVC-lite in less than a second.
- Manual proof in Isabelle/HOL takes around 1 page (completed within a few hours).
Current state of implementation

- Interface with ICS and arithmetic provers which support the SMT-LIB format via oracle, e.g., haRVey, CVC/CVC-Lite, Yices, Sammy, etc. See SMT-COMP’05 website.
- Interface with SAT solvers, *with proof reconstruction* in Isabelle. Currently tested on MiniSAT and zChaff (joint work with Tjark Weber from TU Munich). Now part of Isabelle2005 distribution.
- Interface with haRVey (SMT prover with congruence closure) with proof reconstruction.
ICS and SMT-LIB interface (oracle)

- Syntax translations are mostly trivial for standard arithmetic operator, except the absolute-value operator, which needs to be encoded via if-then-else operator.
- Set related operations (set membership, subset, set intersection, etc) are not supported by ICS and other provers. Explicit encoding needed in this case (not yet implemented).
- For SMT-LIB, need to convert higher-order types to first-order types (multisorted).
Proof reconstruction

- SMT-LIB solvers: “sat solvers + theory reasoning”
  Propositional "model"

SAT-solver \rightarrow Theory reasoning \rightarrow Conflict clause

- Proof reconstruction in two phases: reconstruction of congruence closure proofs and resolution (SAT) proofs
SAT solver interface

SAT-solvers (zChaff, MiniSAT, Berkmin, . . .)

- Powerful method for deciding Boolean Logic
- Input: Conjunctive Normal Form
- Output: Satisfiable or Unsatisfiable

Issues for integration:

- In Isabelle, we want to prove theorems (i.e. valid - not valid)
  ⇒ prove unsatisfiability of the negation
- In Isabelle, formulas are not restricted to Conjunctive Normal Forms
  ⇒ convert to Conjunctive Normal Form
SAT solver interface: outline

Proving an Isabelle theorem using a SAT-solver?

- Use the negation of the theorem
- Convert to Conjunctive Normal Form (with proofs)
- Give the obtained CNF to the SAT-solver
- Get the answer from the SAT-solver
  - SAT: the original formula is not a theorem
  - UNSAT: the original formula is valid, it is a theorem
- Rebuild the proof within Isabelle
Conjunctive Normal Form

Naive CNF (distribution rules)? **Exponential**

Example: \((a_1 \land b_1) \lor \ldots (a_n \land b_n)\) to CNF: \(2^n\) clauses

Definitional CNF: *introduce new variables* to remain **linear**:

\[ \varphi(\psi) \text{ equivalent to } \exists x.(x \equiv \psi) \land \varphi(x) \]

\(x \equiv \psi\) easily translated to CNF. Example: \(\psi\) is \(a \land b \land c\)

\(x \equiv \psi \iff (\neg x \lor a) \land (\neg x \lor b) \land (\neg x \lor c) \land (\neg a \lor \neg b \lor \neg c \lor x)\)
Reconstructing SAT proofs

- The SAT solver (MiniSAT) returns a complete proof trace. That is, all the resolution steps needed to derive the refutation.
- Logically, the reconstruction is simple. The main concern is efficiency since we are dealing with a big number of clauses.
- Some factors to consider:
  - How do we represent and store clauses in Isabelle/HOL?
  - How do we do resolution?
Clause representation

Represent a clause as a sequent, e.g.,

$$[
eg a; b; \neg c] \implies False$$

represents $$(a \lor \neg b \lor c)$$.  

Resolution: given two clauses

$$[a; b; \neg c] \implies False$$ and $$[d; \neg a; e] \implies False$$

first swap the premise of the first clause:

$$[b; \neg c] \implies \neg a$$

and cut this with the second clause.

The resulting theorem:

$$[d; b; \neg c; e] \implies False$$
Reconstructing congruence closure proofs

- Problem: deciding whether an equation follows from other equations. Restrict to the quantifier free equalities with uninterpreted function symbols.
- The difficulty in deciding the equality is essentially in knowing when and where to apply the congruence rule, and using which equations.
- haRVey produces compact proof traces using two "big-step" rules:
  - a derived rule using reflexivity-symmetry-transitivity,
  - and another rule using shallow congruence rule.
Example of a haRVey proof trace

Goal:

\[ f \ a = a \land f \ (g \ b) = f \ a \land f \ (g \ b) = g \ (f \ a) \land g \ b = g \ (g \ a) \implies a = g \ b \]

Proof trace produced by haRVey:

TRANS: \[ g \ b = g \ (g \ a) \land g \ (f \ a) = g \ (g \ a) \land f \ (g \ b) = g \ (f \ a) \land f \ a = f \ (g \ b) = g \ (f \ a) \implies a = g \ b \]

CONGR: \[ f \ a = g \ a \implies g \ (f \ a) = g \ (g \ a) \]

TRANS: \[ g \ (f \ a) = g \ a \land f \ (g \ b) = g \ (f \ a) \land f \ a = f \ (g \ b) \implies f \ a = g \ a \]

CONGR: \[ a = f \ a \implies g \ (f \ a) = g \ a \]
Some experimental results (SAT)

- A few examples ($\approx$ 100-200 clauses):

<table>
<thead>
<tr>
<th>Formula</th>
<th>SAT-time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUZ015-2.006</td>
<td>0.005</td>
<td>2.435</td>
</tr>
<tr>
<td>PUZ016-2.005</td>
<td>0.003</td>
<td>1.158</td>
</tr>
<tr>
<td>PUZ030-2</td>
<td>0.002</td>
<td>0.485</td>
</tr>
<tr>
<td>SYN010-1.005</td>
<td>0.002</td>
<td>0.369</td>
</tr>
<tr>
<td>SYN093-1.002</td>
<td>0.005</td>
<td>0.133</td>
</tr>
<tr>
<td>SYN094-1.005</td>
<td>0.005</td>
<td>0.742</td>
</tr>
<tr>
<td>MSC007-1.008</td>
<td>0.208</td>
<td>11.546</td>
</tr>
</tbody>
</table>

- None can be solved with standard Isabelle tactics
Some experimental results (haRVey)

From SMT-COMP’05 benchmark, category QFUF:

<table>
<thead>
<tr>
<th>Formula</th>
<th>Size</th>
<th>Confl</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ004-size5.smt</td>
<td>82019</td>
<td>149</td>
<td>164.475</td>
</tr>
<tr>
<td>SEQ011-size2.smt</td>
<td>35051</td>
<td>56</td>
<td>17.639</td>
</tr>
<tr>
<td>SEQ015-size2.smt</td>
<td>1828</td>
<td>20</td>
<td>2.040</td>
</tr>
<tr>
<td>SEQ020-size2.smt</td>
<td>37867</td>
<td>47</td>
<td>17.218</td>
</tr>
<tr>
<td>SEQ032-size2.smt</td>
<td>1496</td>
<td>20</td>
<td>1.874</td>
</tr>
<tr>
<td>SEQ042-size2.smt</td>
<td>4624</td>
<td>91</td>
<td>10.848</td>
</tr>
<tr>
<td>SEQ050-size2.smt</td>
<td>3864</td>
<td>140</td>
<td>26.405</td>
</tr>
</tbody>
</table>
Future work

- Large scale case study: FlexRay protocols.
- Use object specification languages (I/O automata, TLA+), possibly also interface with model checkers for the corresponding specification languages.
- Improve efficiency for proof reconstruction
- Proof reconstruction for (linear) arithmetics.
- Standard proof format for SMT solvers?
Related work (1)

Several efforts on combination of deductive tools:

- the Omega prover [Siekmann, et.al.]: proof assistants for mathematics.
- the SAL framework [SRI]: a common specification language; aimed at integrating different model checkers.
- the PROSPER project: integrating formal verification with CAD/CASE tools.
- the CALIFE project: platform for verification based on synchronized products of automatas.
- ...
Related work (2)

Proof reconstruction from external provers:

- interfacing between Coq and rewrite system ELAN [Nguyen, C. Kirchner and H. Kirchner, 2002]
- interfacing between Isabelle and first-order automated provers [Meng, Quigley, Paulson, 2005]
- interfacing Isabelle with zChaff [Tjark Weber, 2005].
- ...