Comparing Proof Systems for Linear Real Arithmetic Using LFSC

Andrew Reynolds
Liana Hadarean

July 15, 2010
• **University of Iowa**
  Andrew Reynolds, Cesare Tinelli, Aaron Stump

• **New York University**
  Liana Hadarean, Yeting Ge, Clark Barrett
Motivation for this work

- SMT solvers are difficult to verify
  - Code may be complex (10k+ loc)
  - Code is subject to change

- Alternatively, solvers can justify answers with proofs

- There is need for third party certification
  - Must ensure that proof is valid
• For “satisfiable”:
  – Provide a satisfying assignment

• For “unsatisfiable”:
  – Provide a proof of unsatisfiability
Architecture

Solver

sat

unsat

Assignment

Proof of Unsatisfiability

Proof Checker

Proof Valid

Proof Invalid
• **Flexibility**
  - Different solvers have different needs
  - Solvers can change over time
  - Many different theories

• **Speed**
  - Practical for use with solvers
  - Measured time against solving time
• Certification of proofs in QF_LRA
  – Use LFSC for proof checking

• Experiments with QF_LRA proof systems
  – Examine declarative vs computational
  – Use CVC3 for proof generation
• Edinburgh Logical Framework (LF) [Harper et al 1993]
  – Based on type theory
  – Meta framework for defining logical systems

• LF with side conditions (LFSC) [Stump et al 2008]
  – Meta-logical proof checker
  – Side Conditions
  – Support for Integer, Rational arithmetic
  – If proof term type-checks,
    Then proof is considered valid
Example proof rule

\[
\begin{array}{c}
\psi_1 \\
\vdash \\
\psi_2 \\
\hline
\psi_1 \land \psi_2
\end{array}
\]

```
(declare_and_intro
  (! f1 formula
  (! f2 formula
  (! p1 (proof f1)
  (! p2 (proof f2)
   (proof (and f1 f2))))))))
```
\[
\begin{array}{c}
\frac{p > 0}{\{p \downarrow c, \ c \not> 0\}}
\end{array}
\]

(declare ineq_contradiction
  (! p poly
   (! pl (proof (> p 0))
     (! s (^ (is_positive (simplify p)) ff) false))))
- Side conditions
  - Written in simply typed functional language
  - Most are concise (less than 10 loc)
Proof rule with side condition

\[
\frac{p > 0}{\{ p \downarrow c, \, c \nleq 0 \}}
\]

(program simplify ((p poly)) real
(match p
  ((poly c' l')
    (match (is_zero l')
      (tt c')
      (ff fail)))))

...(^ (is_positive (simplify p)) ff)
Why side conditions?

- Mirror high-performance solver inferences
- More Efficient
  - Smaller Proof Size
  - Faster Checking time
- Amount can be fine tuned

Fully Declarative ← Fully Computational
• **Incremental Checking**
  – Proof checking occurs while reading proof

• **Deferred Resolution**
  – Efficient to check boolean inferences

• **Compiled Side Condition Code**
  – Compiled instead of interpreted code
• Demonstrate capabilities of LFSC
  – Flexibility in:
    • Handling new logic (QF_LRA)
    • Defining multiple proof systems for this logic
• Developed LFSC signatures for QF_LRA
• Instrumented CVC3 to produce proofs in system
• Comparative analysis
• Refutation based prover for SMT
• Support for many different logics
  – Integer/Real, Arrays, Data types, etc.
  – Support for quantifiers
• Proof generation
  – Native format
• Did not modify CVC3 core
• Translated CVC3 Proofs to LFSC
  – Opportunity to test different translations

CVC3 to LFSC proofs

CVC3

sat

unsat

CVC3 Proof of Unsatisfiability

LFSC Proof of Unsatisfiability

LFSC
• Literal translation (Lit)
  – Mimics the structure of CVC3 proofs

• Liberal translation (Lib)
  – Compacts portions of proof to side conditions
  – Limits compaction to QF_LRA theory lemmas

• Aggressive Liberal translation (Lib-A)
  – Extends compaction to equality reasoning proof fragments
• Proof derives false from:
  – Input formulas
  – *Theory Lemmas*
    • i.e. (x+1 > x)

• Proof Rules
  – Many rules (100+)
  – Rewrite axioms
  – Mostly Declarative
• Theory lemmas in QF_LRA
  – Ex: \( \neg (2x > 2y) \lor \neg (y > x + 5) \)
  – Proof of unsatisfiability from assumptions
• Theory lemmas in QF_LRA
  – Ex: \( \neg (2x > 2y) \lor \neg (y > x + 5) \)
  – Can be done by finding set of coefficients

\[
\begin{align*}
\frac{1}{2}^* & \quad 2x > 2y \\
1^* & \quad y > x + 5 \\
\hline \\
& \quad x + y > y + x + 5 \\
\Downarrow \\
& \quad 0 > 5
\end{align*}
\]
• LFSC proofs use *polynomial* formulas
  – Ex: Instead of $2x > 2y$, $(2x - 2y) \downarrow > 0$

• Proof of theory lemmas are always of the form:

\[
\vdots \quad c_p \sim 0
\]

• Intuition: For each CVC3 rule, determine corresponding coefficient to multiply each premise by to obtain contradictory polynomial $c_p$
• CVC3 rules mapped to polynomial operations
• Applies to all proof rules for theory lemmas
  – However, not applicable to boolean portions
• Compaction occurs because:
  – Condense redundant operations
  – Eliminate trivial subproofs, such as those involving only rewrite axioms
• Theory lemma example:

\[
\begin{align*}
\frac{1}{2}* & \quad 2x > 2y \\
1 & \quad y > x + 5 \\
\hline
x + y & > y + x + 5 \\
\downarrow \\
0 & > 5
\end{align*}
\]
Map to operations on polynomials

\[
\begin{align*}
2x > 2y & \quad 2x > 2y \iff x > y \\
& \quad x > y \\
& \quad y > x + 5 \\
& \quad x > x + 5 \\
\hline
\end{align*}
\]

Map to operations on polynomials

\[
\begin{align*}
(2x - 2y) \downarrow > 0 & \quad 0 = 0 \\
& \quad \frac{1}{2} (2x - 2y) \downarrow > 0 \\
& \quad (\frac{1}{2} (2x - 2y) + (y - (x + 5))) \downarrow > 0 \\
\hline
\end{align*}
\]
\[
\frac{(2x - 2y) \downarrow > 0}{0 = 0}
\]
\[
\frac{1}{2}(2x - 2y) \downarrow > 0
\]
\[
\frac{(y - (x + 5)) \downarrow > 0}{0 = 0}
\]
\[
\frac{(\frac{1}{2}(2x - 2y) + (y - (x + 5))) \downarrow > 0}{0 = 0}
\]
\[
\frac{(\frac{1}{2}(2x - 2y) + (y - (x + 5))) \downarrow > 0}{0 = 0}
\]
\[
\frac{2x - 2y > 0}{x - y > 0}
\]
\[
\frac{-x + y - 5 > 0}{-5 > 0}
\]

Remove redundant operations
• Attempt to compact all theory inferences
• When conversion gets stuck, Switch to literal translation

\[
\begin{align*}
\vdots \\
(x - y) \downarrow > 0 \\
\vdots \\
x > z \quad x > y \\
\vdots \\
x > z \land x > y
\end{align*}
\]

Compact Translation

Literal Translation
• Selection of 145 unsatisfiable QF_LRA benchmarks
  – Each solved ≤ 60s by CVC3
  – Proof generation ≤ 300s

• Configurations
  – CVC3 native proof \( \text{CVC3} \)
  – Literal \( \text{Lit} \)
  – Liberal \( \text{Lib} \)
  – Aggressive Liberal \( \text{Lib-A} \)
Proof size

Lit vs Lib

Lit Proof Size (KB) vs Lib Proof Size (KB)

Lit vs Lib-A

Lit Proof Size (KB) vs Lib-A Proof Size (KB)
Proof checking vs Solving

Solving vs Lit

Solving vs Lib

CVC3 Solve (sec)

Lit Pf Check (sec)

Lib Pf Check (sec)

FLOC 2010 SMT Workshop
• Theory content 11% on average
• For theory heavy benchmarks
  – Lib compresses proof sizes 34%
  – Lib-A compresses proofs sizes 58% (16% overhead on non-theory benchmarks)
• Lib is the most effective method overall with an average compression of 12%
• When isolated to theory component
  – Lib compresses proof sizes factor of 5.24
  – Lib improves proof checking factor of 2.7
• Overall, Lib proof checking is factor of 2.6 faster than solving time
• LFSC is a pragmatic approach to proof checking
  – Efficient
    • Checking times fast w.r.t. solving
  – Trustworthy
    • Small/not complex side condition code
    • Clear definition of trusted components
  – Flexible
    • Signature is separate from checker
    • Effective for different proof systems
Future work

• Demonstrate scalability for QF_LRA
• Integration with CVC4
  – New decision procedures
  – New logics (arrays etc.)
• Public release of LFSC
  – Tool for signature creation
  – LFSC proof generation library
• Interpolant generating proofs
Questions?