Comparing Proof Systems for Linear Real Arithmetic Using LFSC

Andrew Reynolds
Liana Hadarean

July 15, 2010

Acknowledgements

- **University of Iowa**

  Andrew Reynolds, Cesare Tinelli, Aaron Stump

- **New York University**

  Liana Hadarean, Yeting Ge, Clark Barrett

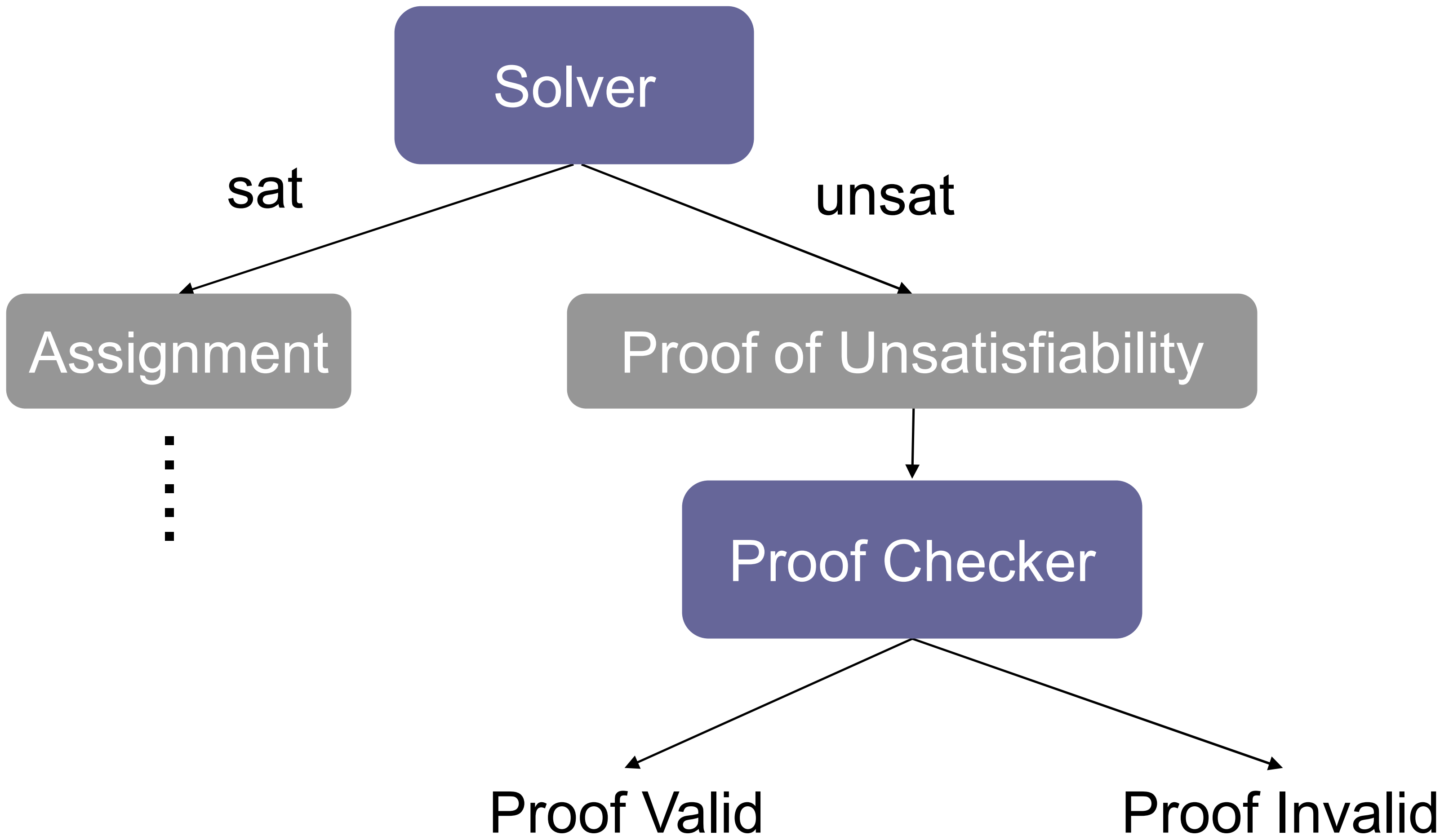
Motivation for this work

- 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

Certifying SMT Solver's Answers

- For "satisfiable":
  - Provide a satisfying assignment
- For "unsatisfiable":
  - Provide a proof of unsatisfiability

Architecture



Proof Checking: Challenges

- **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

Overview

- 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

Proof Checking in LFSC

- 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

$$\frac{\psi_1 \quad \psi_2}{\psi_1 \wedge \psi_2}$$

```
(declare and_intro
     (! f1 formula
     (! f2 formula
     (! p1 (proof f1)
     (! p2 (proof f2)
          (proof (and f1 f2)))))))
```

Proof rule with side condition

$$\frac{p > 0}{\bot}\ \{p \downarrow c,\ c \not> 0\}$$

```
(declare ineq_contradiction
  (! p poly
  (! p1 (proof (> p 0))
  (! s (^ (is_positive (simplify p)) ff)
     false))))
```

Proof rule with side condition

$$\frac{p > 0}{\bot} \quad \{p \downarrow c,\ c \not> 0\}$$

- Side conditions
  - Written in simply typed functional language
  - Most are concise (less than 10 loc)

Proof rule with side condition

$$\frac{p > 0}{\perp}\ \{p \downarrow c,\ c \not> 0\}$$

```
(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

LFSC Optimizations [Oe et al 2009]

- **Incremental Checking**
  - Proof checking occurs while reading proof
- **Deferred Resolution**
  - Efficient to check boolean inferences
- **Compiled Side Condition Code**
  - Compiled instead of interpreted code

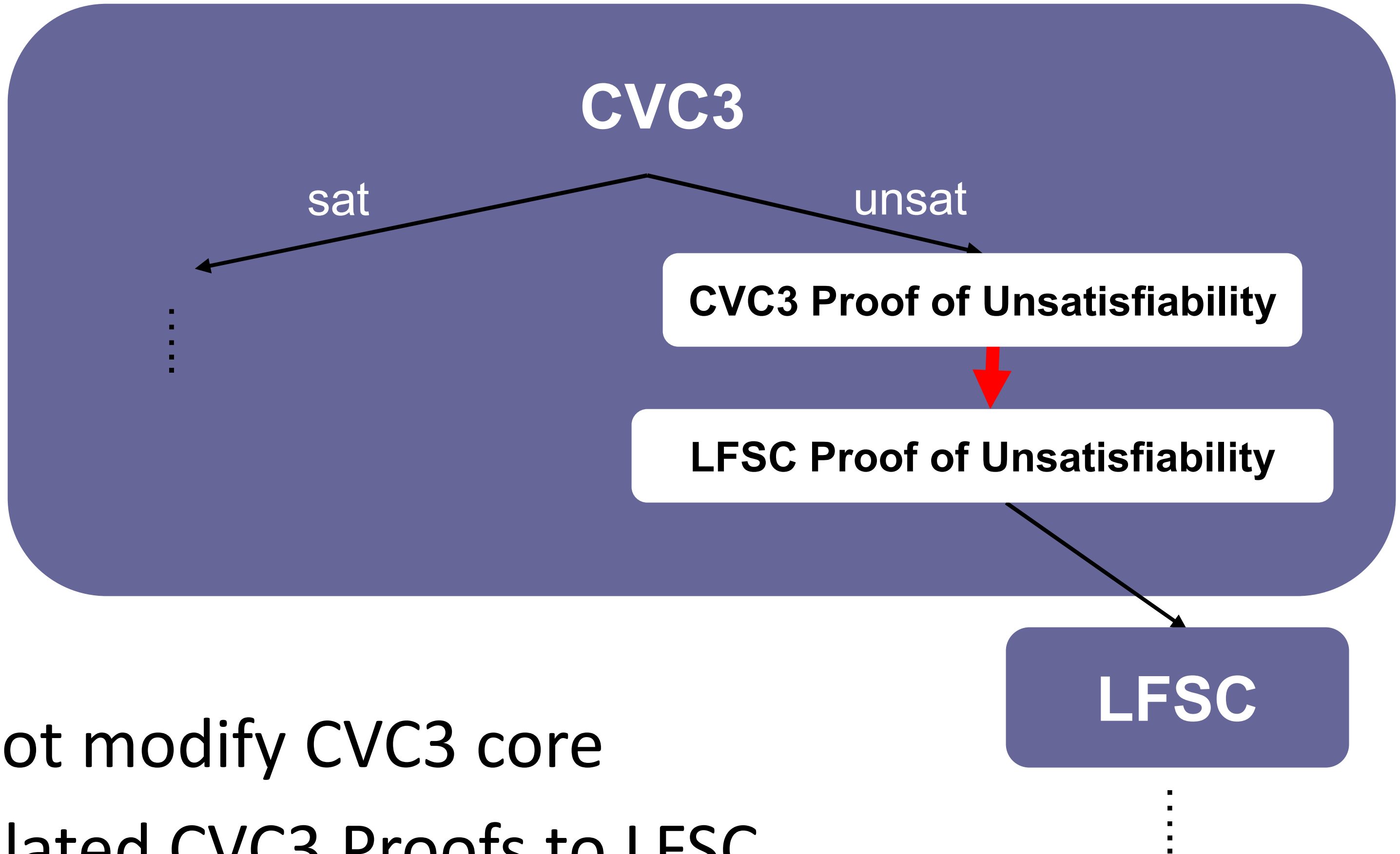
Contributions of this work [2010]

- 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

CVC3

- Refutation based prover for SMT
- Support for many different logics
  - Integer/Real, Arrays, Data types, etc.
  - Support for quantifiers
- Proof generation
  - Native format

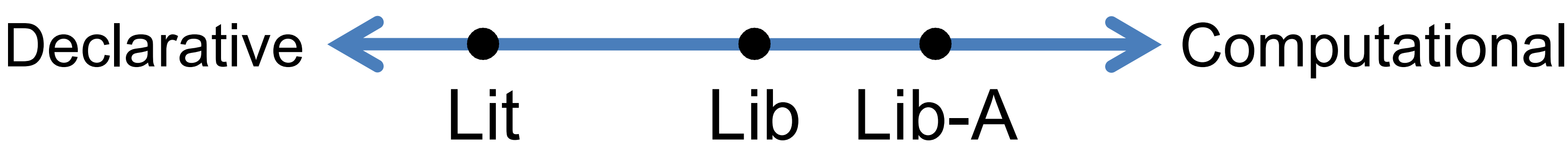
CVC3 to LFSC proofs



- Did not modify CVC3 core
- Translated CVC3 Proofs to LFSC
  - Opportunity to test different translations

Approaches

- 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



CVC3 Proofs

- Proof derives false from:
  - Input formulas
  - *Theory Lemmas*
    - i.e. (x+1 > x)
- Proof Rules
  - Many rules (100+)
  - Rewrite axioms
  - Mostly Declarative

$$\frac{}{\vdash \psi_1 \Leftrightarrow \psi_2}$$

Compaction from CVC3 to LFSC

- Theory lemmas in QF_LRA
  - Ex: $\neg(2x>2y) \vee \neg(y>x+5)$
  - Proof of unsatisfiability from assumptions

Compaction from CVC3 to LFSC

- Theory lemmas in QF_LRA
  - Ex: $\neg(2x>2y) \vee \neg(y>x+5)$
  - Can be done by finding set of coefficients

$$\frac{\begin{array}{rl} \tfrac{1}{2}* & 2x > 2y \\ 1* & y > x + 5 \end{array}}{x + y > y + x + 5}$$

$$\Downarrow$$

$$0 > 5$$

Compaction from CVC3 to LFSC

- LFSC proofs use *polynomial* formulas
  - Ex: Instead of 2x > 2y, (2x – 2y)↓ > 0
- Proof of theory lemmas are always of the form:

$$\frac{\vdots}{c_p \sim 0}$$

- Intuition: For each CVC3 rule, determine corresponding coefficient to multiply each premise by to obtain contradictory polynomial c_p

Compaction from CVC3 to LFSC

- 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

Proof Compaction Example

- Theory lemma example:

$$\frac{\begin{array}{ll} \tfrac{1}{2} * & 2x > 2y \\ 1 * & y > x + 5 \end{array}}{x + y > y + x + 5}$$

$$\Downarrow$$

$$0 > 5$$

Proof Compaction step 1

$$\dfrac{\dfrac{\dfrac{2x > 2y \qquad \overline{2x > 2y \Leftrightarrow x > y}}{x > y} \qquad y > x + 5}{x > x + 5} \qquad \dfrac{\vdots}{x > x + 5 \Leftrightarrow \bot}}{\bot}$$

Map to operations on polynomials

$$\dfrac{\dfrac{\dfrac{(2x - 2y) \downarrow > 0 \qquad \overline{0 = 0}}{\frac{1}{2}(2x - 2y) \downarrow > 0} \qquad (y - (x + 5)) \downarrow > 0}{(\frac{1}{2}(2x - 2y) + (y - (x + 5))) \downarrow > 0} \qquad \dfrac{\vdots}{0 = 0}}{(\frac{1}{2}(2x - 2y) + (y - (x + 5))) \downarrow > 0}$$

Proof Compaction step 2

$$\dfrac{\dfrac{\dfrac{(2x-2y)\downarrow > 0 \qquad \overline{0=0}}{\frac{1}{2}(2x-2y)\downarrow > 0} \qquad (y-(x+5))\downarrow > 0}{(\frac{1}{2}(2x-2y)+(y-(x+5)))\downarrow > 0} \qquad \dfrac{\vdots}{0=0}}{(\frac{1}{2}(2x-2y)+(y-(x+5)))\downarrow > 0}$$

Remove redundant operations

$$\dfrac{\dfrac{2x-2y>0}{x-y>0} \qquad -x+y-5>0}{-5>0}$$

Aggressive Liberal translation

- Attempt to compact all theory inferences
- When conversion gets stuck,

  Switch to literal translation

$$\frac{\vdots}{(x - y) \downarrow > 0}$$

Compact Translation

$$\dfrac{x > z \qquad \dfrac{\vdots}{x > y}}{x > z \wedge x > y}$$

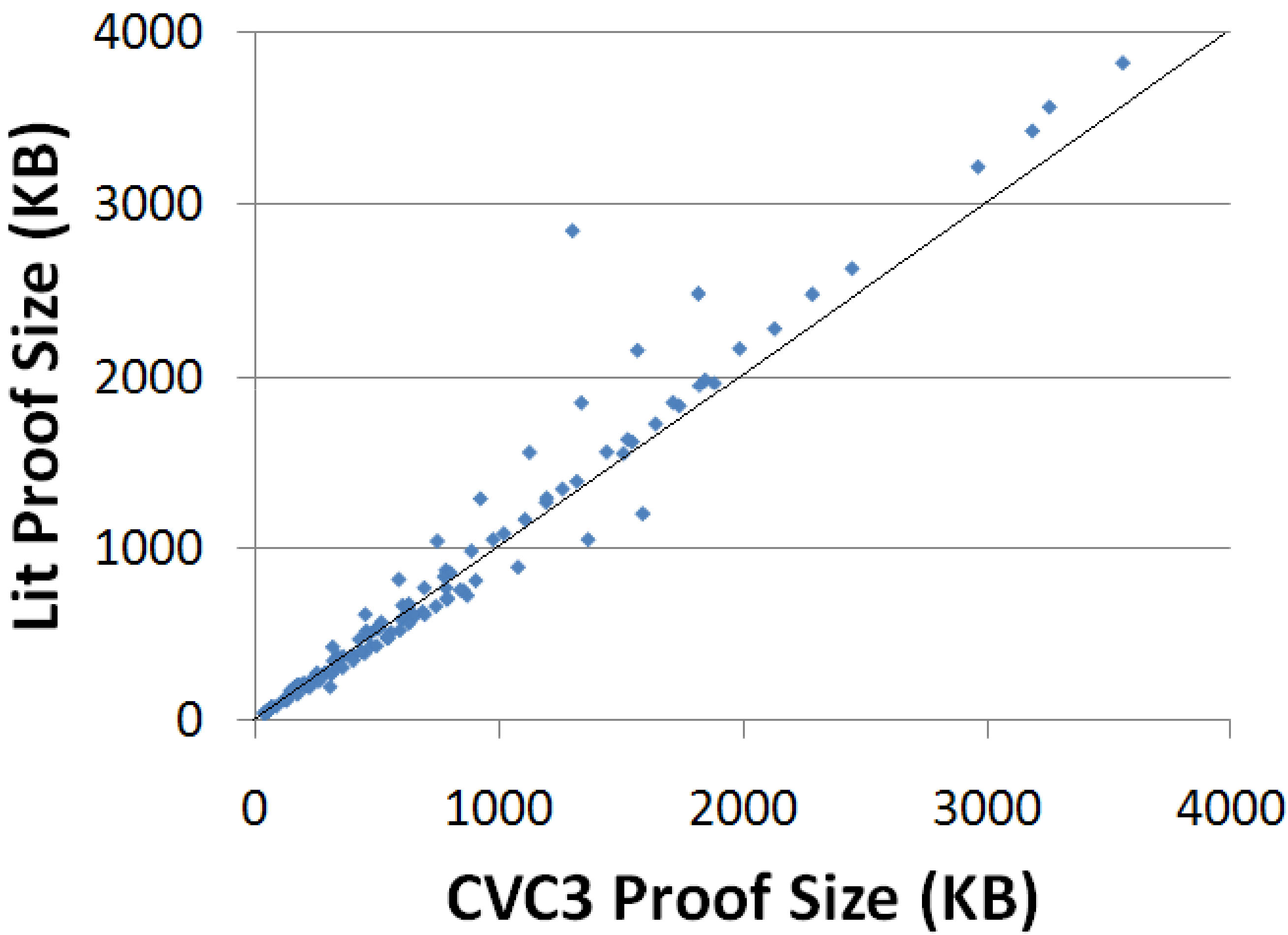
Literal Translation

Experimental results

- Selection of 145 unsatisfiable QF_LRA benchmarks
  - Each solved ≤ 60s by CVC3
  - Proof generation ≤ 300s

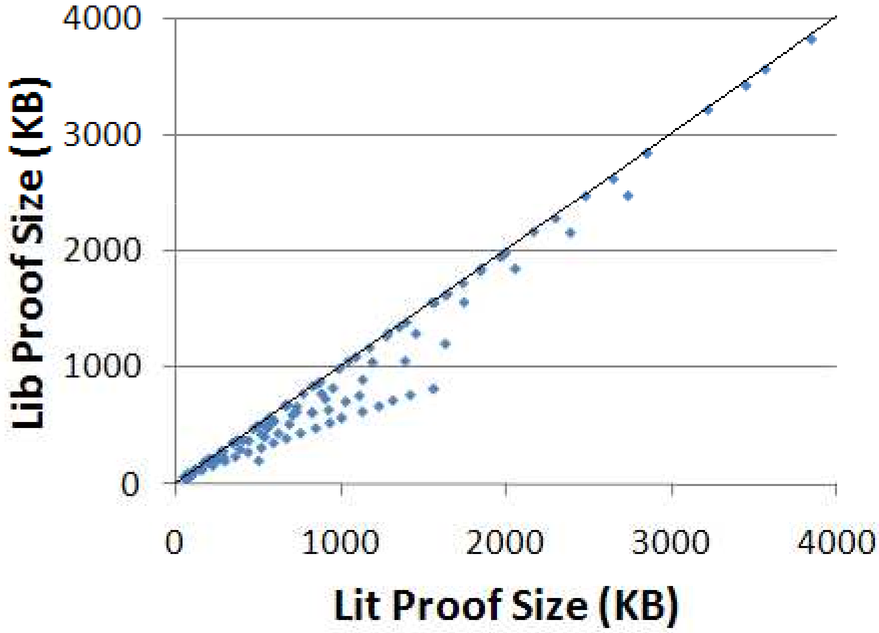
- Configurations
  - CVC3 native proof (**CVC3**)
  - Literal (**Lit**)
  - Liberal (**Lib**)
  - Aggressive Liberal (**Lib-A**)

Proof size CVC3 vs Lit

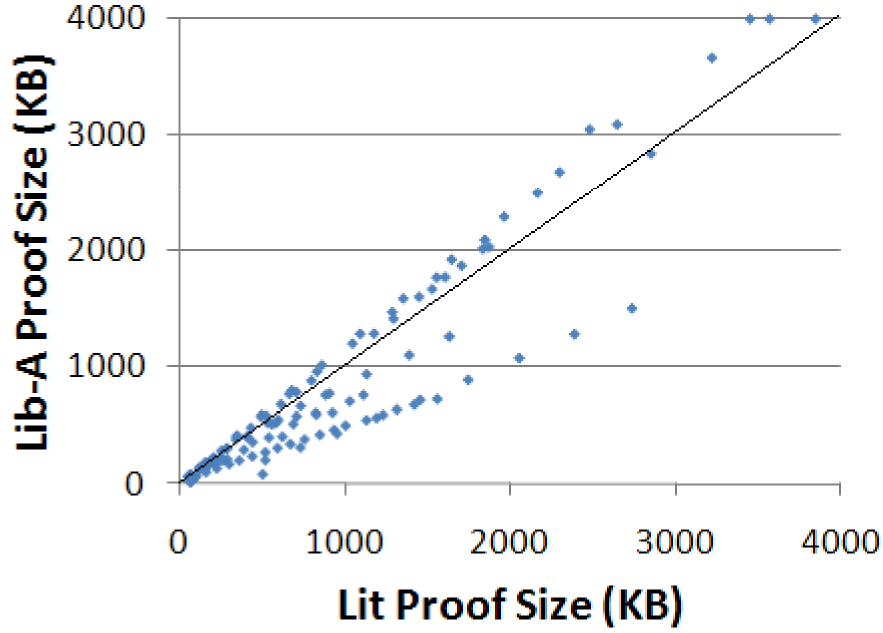


Proof size

Lit vs Lib

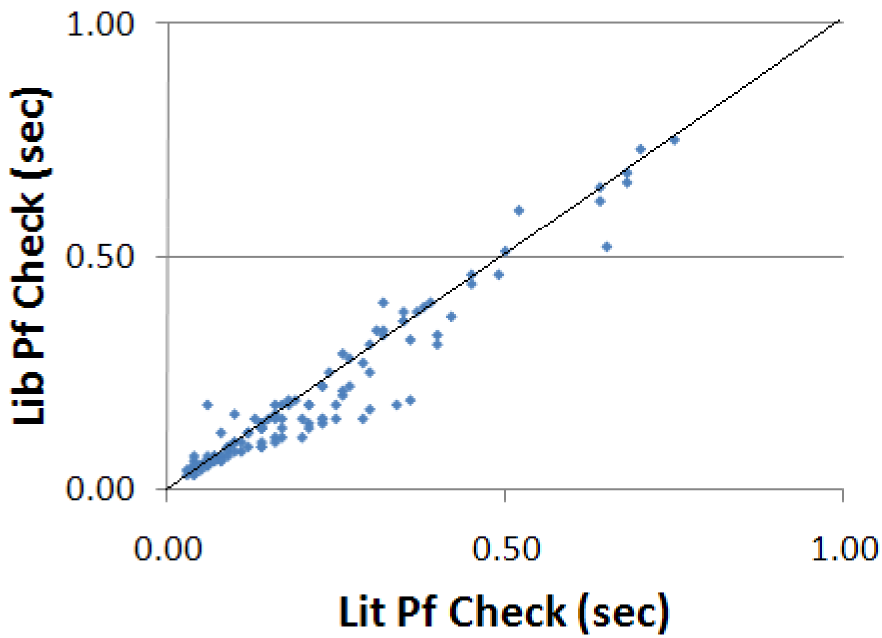


Lit vs Lib-A

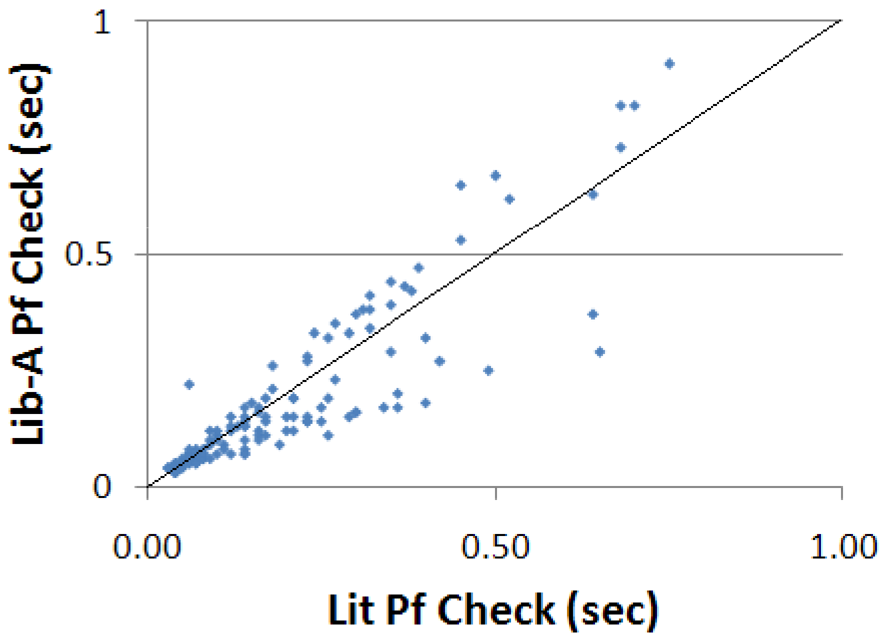


Proof checking time

Lit vs Lib

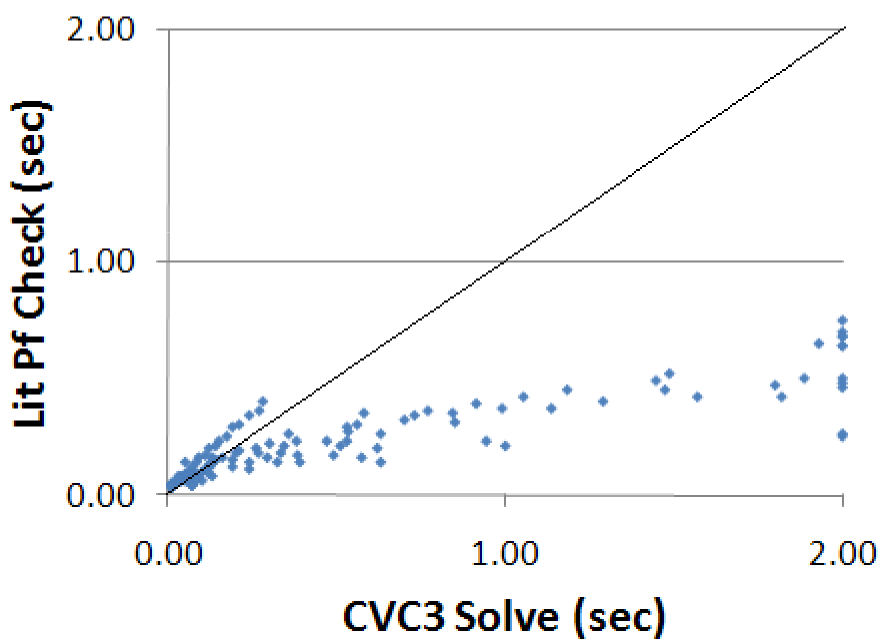


Lit vs Lib-A

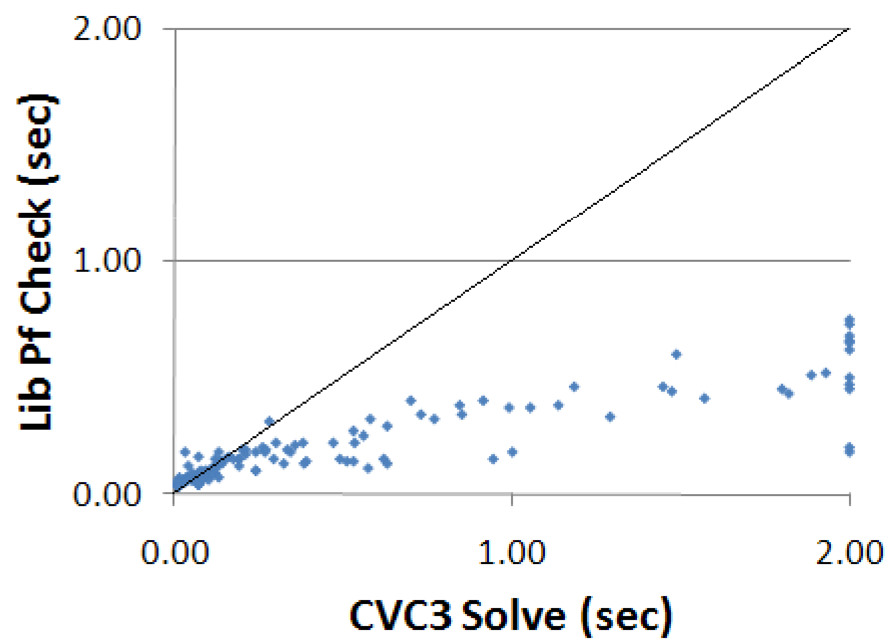


Proof checking vs Solving

Solving vs Lit



Solving vs Lib



Analysis

- 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%

Analysis continued

- 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

Conclusions

- 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?