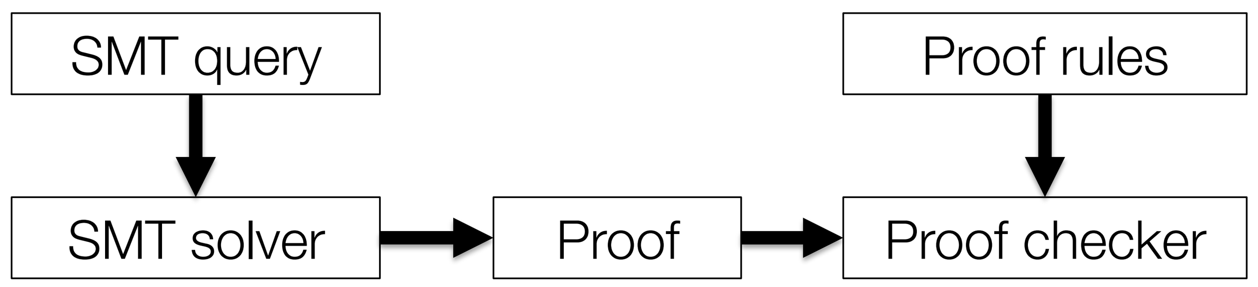
Proofs for Preprocessing in SMT Solvers

Andres Nötzli, Stanford University
`noetzli@stanford.edu`

Background

- Use cases for SMT solvers include verification → need to be able to trust output
- SMT solvers are complex and no complex software is bug free
- Increase confidence in answer by providing an independently checkable proof:



Proof describes reasoning, proof checker makes sure that reasoning is consistent with proof rules

Motivation

- Preprocessing simplifies formulas
- All SMT solvers rely on preprocessing for good performance (and sometimes correctness)
- SMT solvers produce proofs for core procedures but not preprocessing steps
- Manual implementation is tedious and error-prone:
  - Hundreds of rules
  - Solver has to produce proof for each rule
  - Proof checker has to be able to check all rules

Stanford University

Idea

- Most of the preprocessing module of a solver can be expressed as a set of rewrite rules:

- Use a domain-specific language for rewrite rules
- Implement a compiler that:
  - Generates code to perform the rewrite including code to produce a proof
  - Generates proof rule for the proof checker
  - Supports reasoning about rewrite rules

Example

Rewrite rule

```
Name: writeOverRead
(store #a #i (select #a #i)) => #a
```

→ C++ code performing the rewrite

```
if (node[0] == node[2][0] && node[1] == node[2][1] &&
    node.getKind() == kind::STORE &&
    node[2].getKind() == kind::SELECT) {
        return RewriteResponse(REWRITE_DONE, node[0]);
}
```

→ Logical Framework with Side Conditions (LFSC) proof rule

```
(declare wor
  (! s1 sort
  (! s2 sort
  (! i (term s1)
  (! oa (term (Array s1 s2))
  (! a (term (Array s1 s2))
  (! u (th_holds (= _ oa
    (apply _ _ (apply _ _ (apply _ _ (write s1 s2) a) i)
               (apply _ _ (apply _ _ (read s1 s2) a) i))))
  (th_holds (= _ oa a))))))))))
```

→ Verification of rewrite rule

The Domain-Specific Language

- Design goals: intuitive but expressive enough for most rewrite rules
- Syntax based on SMT-LIB syntax for familiarity
- Rules consist of a source template, a target template and a condition (optional)
- Source template: pattern that SMT solver is searching for
- Condition: evaluated at runtime by SMT solver
- Expression is replaced to match target template if source template matches and condition is fulfilled

Reasoning About Rewrite Rules

- High-level description simplifies reasoning
- Verify correctness of single rewrite
  - Automatically: Use SMT solver without processing
  - Semi-automatically: Generate parts of proof for a proof assistant
- Reason about sets of rewrite rules, e.g. find rewrite loops

Implementation

- Currently targeting CVC4, which uses the LFSC meta-logic for proofs and proof rules
- Challenges:
  - Code that performs the rewrites needs to be efficient → optimize across multiple rewrite rules
  - Proofs need to be simple to produce and efficient to check