Leveraging SMT:
Using SMT Solvers to Improve Verification;
Using Verification to Improve SMT Solvers

Chris Conway

New York University

Usable Verification 2010

with
Clark Barrett
Progress in SMT

- Performance has quickly improved since the first SMT-COMP (2005)
- More than a dozen tools available, with varying feature sets
- Richer theories
  - Bit vectors
  - Nonlinear arithmetic
  - Complete fragments of FOL
- Improvements in theory combination
  - Polite theories
  - Sharing via care graphs
- New features
  - Proofs
  - Interpolants
- Improved standardization via SMT-LIB format
Modern SMT solvers have power that is not being tapped
d. E.g., in CAV 2010, we presented a verification procedure using

- Bit vectors
- Arrays
- Uninterpreted functions
- Inductive datatypes
- Quantifiers

You can’t do that with SAT!
SMT Can Benefit from Verification

- New benchmarks lead to:
  - Better heuristics
  - Interesting fragments
  - Improved algorithms

- User demand leads to new features

- You have to tell us where it hurts before you can get your medicine.
SMT Solvers are Closed

- Most SMT solvers are closed source, including some of the best
- In comparison, many SAT solvers release source
- Slows progress by duplicating effort
- Maybe the world doesn’t need a dozen competitive solvers...
SMT-LIB is written to be simple, consistent, and easy to parse; not necessarily easy to read or write.

Logic must be declared, not inferred.

E.g., problems in QF_RDL must have the syntax:

“Closed quantifier-free formulas with atoms of the form:

- \( p \)
- \( (op (− x y) c) \),
- \( (op x y) \),
- \( (op (− (+(x ... x) +(y ... y))) c) \) with \( n > 0 \) occurrences of \( x \) and \( y \), where ...”
Quantifiers Are Still a Problem

Quantifier performance depends on

- User-provided triggers (a black art)
- Counter-intuitive heuristics
- Unreliable detection of complete fragments

```cvc
CVC> T : TYPE = SUBTYPE( LAMBDA( x : INT ) : x > 0 );
*** Fatal exception:
Type Checking error: Unable to prove witness for subtype:

    SUBTYPE((LAMBDA (x: INT): (x > 0)))

The failed condition is:

    (EXISTS (_cvc3_: INT) : (0 <= (-1 + _cvc3_)))
```
Managing the SMT interface

Glue code is non-trivial

- Encoding program/specification into SMT instance
- Decoding counterexamples to program/specification-level expressions

\[ Mux-Sem: \]

\[ y: \text{integer where } y = 1 \]

\[ \|_{i=1}^N P[i] :: \]

\[ \ell_0: \text{loop forever do} \]

\[ \ell_1: \text{Non-critical} \]

\[ \ell_2: \text{request } y \]

\[ \ell_3: \text{Critical} \]

\[ \ell_4: \text{release } y \]

Invariant:

\[ \forall i, j : i \neq j \land at_{\ell_3}[i] \implies \neg at_{\ell_3}[j] \]
$ java MuxSemNProver
<...>

Process counter: pi_0

Label identifiers: {l_0=0, l_1=1, l_2=2, l_3=3, l_4=4, label(lbl)=5, label(lbl_0)=6}

Apply rule BINV to: ((NOT (i = j) AND ((pi_0[i] = 3))) => NOT ((pi_0[j] = 3)))

Invariant is not preserved by the transition relation.

Counterexample:
NOT (FORALL (_BD1TY1: SUBTYPE((LAMBDA (n_4: INT): ((1 <= n_4) AND (_N >= n_4)))), _BD2TY2: SUBTYPE((LAMBDA (n_5: INT): ((1 <= n_5) AND (_N >= n_5)))),
(pi_0'[_BD1TY1] = 3)
(pi_0'[_BD2TY2] = 3)
NOT (SKOLEM_10171 = SKOLEM_10170)
(0 <= SKOLEM_10171)
NOT (0 <= (-1 + (-1 * SKOLEM_10171)))
(0 <= (1 + (-1 * SKOLEM_10171) + _N))
(0 <= SKOLEM_10170)
NOT (0 <= (-1 + (-1 * SKOLEM_10170)))
(0 <= (1 + (-1 * SKOLEM_10170) + _N))
NOT (0 <= (-2 + _y))
NOT (0 <= (-1 + (-1 * _y)))
(0 <= _P_i)
NOT (0 <= (-1 + (-1 * _P_i)))
(0 <= (1 + (-1 * _P_i) + _N))
NOT ((pi_0 WITH [__P_i] := 0) = pi_0')
NOT (SKOLEM_454662 = pi_0'[SKOLEM_454644])
NOT ((pi_0 WITH [__P_i] := 1) = pi_0')
NOT (SKOLEM_516628 = pi_0'[SKOLEM_516611])
NOT (pi_0' = pi_0)
NOT (0 <= (-1 + (-1 * pi_0'[SKOLEM_519502])))
NOT (0 <= (-1 + (-1 * pi_0'[SKOLEM_519502])))
(0 <= (1 + _y))
NOT ((pi_0 WITH [__P_i] := 2) = pi_0')
NOT (SKOLEM_691320 = pi_0'[SKOLEM_691302])

FORALL (_BD1TY1: SUBTYPE((LAMBDA (n_4: INT): ((1 <= n_4) AND (_N >= n_4)))), _BD2TY2: SUBTYPE((LAMBDA (n_5: INT): ((1 <= n_5) AND (_N >= n_5)))),
(pi_0'[_BD1TY1] = 3)
(pi_0'[_BD2TY2] = 3)
NOT (SKOLEM_10171 = SKOLEM_10170)
(0 <= SKOLEM_10171)
NOT (0 <= (-1 + (-1 * SKOLEM_10171)))
(0 <= (1 + (-1 * SKOLEM_10171) + _N))
(0 <= SKOLEM_10170)
NOT (0 <= (-1 + (-1 * SKOLEM_10170)))
(0 <= (1 + (-1 * SKOLEM_10170) + _N))
NOT (0 <= (-2 + _y))
NOT (0 <= (-1 + (-1 * _y)))
(0 <= _P_i)
NOT (0 <= (-1 + (-1 * _P_i)))
(0 <= (1 + (-1 * _P_i) + _N))
NOT ((pi_0 WITH [__P_i] := 0) = pi_0')
NOT (SKOLEM_454662 = pi_0'[SKOLEM_454644])
NOT ((pi_0 WITH [__P_i] := 1) = pi_0')
NOT (SKOLEM_516628 = pi_0'[SKOLEM_516611])
NOT (pi_0' = pi_0)
NOT (0 <= (-1 + (-1 * pi_0'[SKOLEM_519502])))
NOT (0 <= (-1 + (-1 * pi_0'[SKOLEM_519502])))
(0 <= (1 + _y))
NOT ((pi_0 WITH [__P_i] := 2) = pi_0')
NOT (SKOLEM_691320 = pi_0'[SKOLEM_691302])
$ java MuxSemNProver
<...>

Process counter: pi_0

Label identifiers: {l_0=0, l_1=1, l_2=2, l_3=3, l_4=4, label(lbl)=5, label(lbl_0)=6}

Apply rule BINV to: ((NOT (i = j) AND ((pi_0[i] = 3))) => NOT ((pi_0[j] = 3)))

Invariant is not preserved by the transition relation.

Counterexample:
 NOT (FORALL (_BD1TY1: SUBTYPE((LAMBDA (n_4: INT): ((1 <= n_4) AND (_N >= n_4)))), _BD2TY2: SUBTYPE((LAMBDA (n_4: INT): ((1 <= n_4) AND (_N >= n_4)))),
 (pi_0'[SKOLEM_10171] = 3)
 (pi_0'[SKOLEM_10170] = 3)
 NOT (SKOLEM_10171 = SKOLEM_10170)
 (0 <= SKOLEM_10171)
 NOT (0 <= (-1 + (-1 * SKOLEM_10171)))
 (0 <= (1 + (-1 * SKOLEM_10171) + _N))
 (0 <= SKOLEM_10170)
 NOT (0 <= (-1 + (-1 * SKOLEM_10170)))
 (0 <= (1 + (-1 * SKOLEM_10170) + _N))
 NOT (0 <= (-2 + _y))
 NOT (0 <= (-1 + (-1 * _y)))
 (0 <= _P_i)
 NOT (0 <= (-1 + (-1 * _P_i)))
 (0 <= (1 + (-1 * _P_i) + _N))
 NOT ((pi_0 WITH [__P_i] := 0) = pi_0')
 NOT (SKOLEM_454662 = pi_0'[SKOLEM_454644])
 NOT ((pi_0 WITH [__P_i] := 1) = pi_0')
 NOT (SKOLEM_516628 = pi_0'[SKOLEM_516611])
 NOT (pi_0' = pi_0)
 NOT (0 <= (-1 + (-1 * pi_0'[SKOLEM_519502])))
 NOT (0 <= (-1 + (-1 * pi_0[SKOLEM_519502])))
 (0 <= (1 + _y))
 NOT ((pi_0 WITH [__P_i] := 2) = pi_0')
 NOT (SKOLEM_691320 = pi_0'[SKOLEM_691302])

\[ at_{l_2}[1], at_{l_3}[2], y = 1 \]
$ java MuxSemNProver
<br...>

Process counter: \( \pi_0 \)

Label identifiers: \( \{ l_0=0, l_1=1, l_2=2, l_3=3, l_4=4, \text{label(lbl)}=5, \text{label(lbl0)}=6 \} \)

Apply rule BINV to: \((\text{NOT } (i = j) \land (\pi_0[i] = 3)) \implies \text{NOT } (\pi_0[j] = 3))\)

Invariant is not preserved by the transition relation.

Counterexample:
\[
\begin{align*}
\text{NOT } & (\text{FORALL } (_\text{BD1TY1}: \text{SUBTYPE}((\text{LAMBDA } (n_4: \text{INT}): ((1 <= n_4) \land (\_N >= n_4)))), _\text{BD2TY2}: \text{SUBTYPE}((\text{LAMBDA } (n_5: \text{INT}): ((1 <= n_5) \land (\_N >= n_5))])))
\end{align*}
\]
\[
\text{FORALL } (_\text{BD1TY1}: \text{SUBTYPE}((\text{LAMBDA } (n_4: \text{INT}): ((1 <= n_4) \land (\_N >= n_4)))), _\text{BD2TY2}: \text{SUBTYPE}((\text{LAMBDA } (n_5: \text{INT}): ((1 <= n_5) \land (\_N >= n_5)))))
\]
\[
\text{pi}_0'[\text{SKOLEM}_{10171}] = 3
\]
\[
\text{pi}_0'[\text{SKOLEM}_{10170}] = 3
\]
\[
\text{NOT } (\text{SKOLEM}_{10171} = \text{SKOLEM}_{10170})
\]
\[
(0 <= \text{SKOLEM}_{10171})
\]
\[
(\text{NOT } (0 <= (1 + (-1 * \text{SKOLEM}_{10171}) + \_\text{N})))
\]
\[
(0 <= \text{SKOLEM}_{10170})
\]
\[
(\text{NOT } (0 <= (1 + (-1 * \text{SKOLEM}_{10170}) + \_\text{N})))
\]
\[
(0 <= \text{SKOLEM}_{10171})
\]
\[
(\text{NOT } (0 <= (-2 + \_y)))
\]
\[
(\text{NOT } (0 <= (-1 + (-1 * \_y))))
\]
\[
(0 <= \_\text{P}_i)
\]
\[
(\text{NOT } (0 <= (1 + (-1 * \_\text{P}_i) + \_\text{N})))
\]
\[
(\text{NOT } ((\pi_0 \text{ WITH } [\_\text{P}_i] := 0)) = \pi_0')
\]
\[
(\text{NOT } (\text{SKOLEM}_{454662} = \pi_0'[\text{SKOLEM}_{454644}])
\]
\[
(\text{NOT } ((\pi_0 \text{ WITH } [\_\text{P}_i] := 1)) = \pi_0')
\]
\[
(\text{NOT } (\text{SKOLEM}_{516628} = \pi_0'[\text{SKOLEM}_{516611}])
\]
\[
(\text{NOT } (\pi_0' = \pi_0])
\]
\[
(\text{NOT } (0 <= (1 + \_y)))
\]
\[
(\text{NOT } ((\pi_0 \text{ WITH } [\_\text{P}_i] := 2)) = \pi_0')
\]
\[
(\text{NOT } (\text{SKOLEM}_{691320} = \pi_0'[\text{SKOLEM}_{691302}])
\]

at\_l_2'[1], at\_l_3'[2], y = 1

at\_l_3'[1], at\_l_3'[2], y' = 0
Summary

- If you’re an SMT user, think about pushing the tool harder
- If you’ve got hard problems, let us know
- If you’ve got ideas to improve SMT, consider contributing to an existing project
- If you’ve got glue code, maybe we should share?