SMT Solvers: New Oracles for the HOL Theorem Prover

Tjark Weber

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

ARG Lunch
17 November 2009
HOL4 is a popular interactive theorem prover.

Interactive theorem proving needs **automation**.
HOL4 is a popular interactive theorem prover.

Interactive theorem proving needs **automation**.

→ Use SMT solvers to decide SMT formulas.
SMT Solvers: New Oracles for the HOL Theorem Prover

Tjark Weber

**System Overview**

- Input formula
- Negation, translation
- Error message
- Theorem
- SMT–LIB or native format
- Satisfiable?
  - Yes
  - No

**HOL4**
Polymorphic $\lambda$-calculus, based on Church’s simple theory of types:

- $\sigma ::= \alpha | (\sigma_1, \ldots, \sigma_n)c$
- $t ::= x_\sigma | c_\sigma | (t_{\sigma \rightarrow \tau} \cdot t_\sigma)_\tau | (\lambda x_\sigma \cdot t_\tau)_{\sigma \rightarrow \tau}$
Polymorphic \( \lambda \)-calculus, based on Church’s simple theory of types:

\[
\begin{align*}
\sigma & ::= \alpha \mid (\sigma_1, \ldots, \sigma_n) c \\
t & ::= x_\sigma \mid c_\sigma \mid (t_{\sigma \rightarrow \tau} t_\sigma)_\tau \mid (\lambda x_\sigma . t_\tau)_{\sigma \rightarrow \tau}
\end{align*}
\]

Extensive libraries:

- quantifiers (of arbitrary order)
- arithmetic (\texttt{nat}, \texttt{int}, \texttt{real}, \ldots)
- data types (tuples, records, bit vectors, \ldots)

\(\implies\) much of mathematics and computer science
Goal: To decide the satisfiability of (quantifier-free) first-order formulas with respect to combinations of (decidable) background theories.

\[ \varphi ::= \mathcal{A} \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \]
Satisfiability Modulo Theories: Example

Theories:
- $\mathcal{I}$: theory of integers
  \[ \Sigma_\mathcal{I} = \{ \leq, +, -, 0, 1 \} \]
- $\mathcal{L}$: theory of lists
  \[ \Sigma_\mathcal{L} = \{ =, \text{hd}, \text{tl}, \text{nil}, \text{cons} \} \]
- $\mathcal{E}$: theory of equality
  \[ \Sigma: \text{free function and predicate symbols} \]

Problem: Is
\[ x \leq y \land y \leq x + \text{hd}(\text{cons} \ 0 \ \text{nil}) \land P(f\ x - f\ y) \land \neg P\ 0 \]
satisfiable in $\mathcal{I} \cup \mathcal{L} \cup \mathcal{E}$?
We must translate HOL formulas into the input language of SMT solvers.

1. SMT-LIB format
2. Yices’s native format
SMT-LIB is the **standard input format** for SMT solvers.

- LISP-like syntax
- Based on first-order logic
- Modular: different “theories” and “logics”
- [http://goedel.cs.uiowa.edu/smtlib/](http://goedel.cs.uiowa.edu/smtlib/)
Yices is a competitive SMT solver. It supports both SMT-LIB and a native input format.

- LISP-like syntax
- Based on higher-order logic
- Supports data types, tuples, records, $\lambda$-expressions
## Features: SMT-LIB vs. Yices

<table>
<thead>
<tr>
<th>Feature</th>
<th>SMT-LIB</th>
<th>Yices</th>
<th>SMT-LIB</th>
<th>Yices</th>
</tr>
</thead>
<tbody>
<tr>
<td>int, real</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
<td>✓</td>
</tr>
<tr>
<td>nat, bool, →</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>prop. logic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>equality</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FOL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HOL</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>arithmetic</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Recursion & Abstraction

We translate HOL formulas by recursion over their term structure. Abstraction is used to deal with unsupported terms/types.

Example: $P_{\alpha \rightarrow \text{bool}} \times_{\alpha}$

**SMT-LIB**
- :extrasorts (a)
- :extrafuns ((x a))
- :extrapreds ((P a))
- :formula (not (P x))

**Yices**
- (define-type a)
- (define P::(-> a bool))
- (define x::a)
- (assert (not (P x)))
Propositional Logic

A simple dictionary approach is sufficient for many HOL4 constants.

- \( T, F, \iff, \implies, \lor, \land \) and \( \neg \)
- \( = \)
- if \( c \) then \( t_1 \) else \( t_2 \) and bool_case \( t_1 \ t_2 \ c \)

SMT-LIB makes a clear distinction between terms and formulas.
SMT-LIB/Yices support directly:

- Types `int`, `real`, and (Yices only) `nat`
- Numerals (e.g., `3.14`)
- Negation, addition, subtraction, multiplication
- Comparison operators `<`, `≤`, `>`, `≥`
For certain other HOL4 functions, e.g., \texttt{min}, \texttt{max} and \texttt{abs}, we introduce suitable definitions.

Example: \( \texttt{abs } x_{\text{int}} \geq 0 \)

\begin{verbatim}
:extrafuns ((hol_int_abs Int Int) (x Int))
:assumption (forall (?x Int)
  (= (hol_int_abs ?x)
    (ite (< ?x 0) (- 0 ?x) ?x)))
:formula (not (>= (hol_int(abs x) 0))
\end{verbatim}
Let Expressions

SMT-LIB allows let expressions only in formulas (but not in terms). We translate the former and eliminate the latter.

Example: \( \text{let } x = 1 \text{ in } x > 0 \)

:formula (not (let (?x 1) (> ?x 0)))

In contrast, Yices allows let expressions to occur anywhere.
Quantifiers

SMT-LIB supports first-order quantification. Higher-order quantification is abstracted away.

Yices supports universal and existential quantifiers of arbitrary order.

Example: $\forall f_{\alpha \rightarrow \beta} \exists g_{\beta \rightarrow \alpha} \forall x_\alpha. g(f\ x) = x$

(define-type a)
(define-type b)
(assert (not (forall (f::(-> a b))
  (exists (g::(-> b a))
    (forall (x::a) (= (g (f x)) x))))))
Yices provides a lambda construct, which is used to translate \( \lambda \)-abstractions. We first perform \( \beta \)-normalization and \( \eta \)-expansion in HOL4.

Functions of more than one argument are curried.

Function update \((a \rightarrow b) f\) becomes \(\text{update } f \ a \ b\).
Tuples

Product types $\alpha \times \beta$ are mapped to their Yices counterparts, tuple a b.

HOL4’s comma operator, $(x, y)$, is translated as mk-tuple x y.

Accessor functions for a tuple’s components, FST p and SND p, are translated as select p 1 and select p 2, respectively.

Tuples with more than two components are supported through nesting.
Records

Record types in HOL4 are semantically equivalent to product types, but with named field access and update.

Example:

```
Hol_datatype 'person = < | employed : bool ; age : num | >`
```

```
(define-type person
 (record employed::bool age::nat))
```

- Field selection `x.age`: `select x age`
- Field update `x with employed := e`: `update x employed e`
- Record literals, e.g., `< | employed := F ; age := 65 | >`: syntactic sugar for a sequence of field updates
In HOL4, record types can depend on type arguments. Since Yices only supports monomorphic types, we may need to create multiple copies of a polymorphic record type.

Example: `Hol_datatype 'foo = < | bar : 'a | >`

An occurrence of both $(\alpha)foo$ and $(\beta)foo$ in the input formula leads to two type definitions

```
(define-type a)
(define-type foo1 (record bar1::a))
(define-type b)
(define-type foo2 (record bar2::b))
```
Yices supports recursive data types.

Example: `Hol_datatype 'list = NIL | CONS of 'a => list`

```
(define-type a)
(define-type list (datatype NIL
  (CONS hd::a tl::list)))
```

- **Monomorphisation**, just like for record types
- Case distinction uses Yices’s **recognizers**: e.g., `list_case b f l` becomes `ite (NIL? l) b (f (hd l) (tl l))`. 
Bit Vectors (I)

Fixed-width bit-vector types, e.g., `word8`, `word32`, are translated to their Yices counterparts: as `bitvector 8`, `bitvector 32`, etc.

Yices supports directly:

- Bit-vector literals
- Concatenation, extraction, shift
- Bitwise logical operations
- Addition, subtraction, multiplication, two’s complement
- Signed and unsigned comparison
HOL4’s \texttt{w2w} function is translated using either \texttt{bv-extract} or \texttt{bv-concat}, depending on the width of its argument and result.

Extracting a single bit from a bit vector, denoted by \texttt{'} in HOL4, is translated using Yices’s \texttt{bv-extract} function.
The translation is *soundness critical*: bugs could lead to inconsistent theorems in HOL4.

Therefore, it is important to get every detail right.
Uniformly generating fresh identifiers is easier than re-using HOL4 identifiers:

- Identifiers must not clash with interpreted functions or keywords that have special meaning to the SMT solver.
- Identifiers must not contain invalid characters.
- Generated identifiers must be distinct from each other.
There are subtle semantic differences between certain HOL4 and (allegedly corresponding) SMT-LIB/Yices functions.

- Subtraction \( m - n \) on naturals:
  
  \[
  \text{(define hol_num_minus::(-> nat nat nat)}\]  
  \[
  \text{(lambda (x::nat y::nat)}\]  
  \[
  \text{(ite (< x y) 0 (- x y))}}\]

- \( x \text{ div } 0 \) and \( x \text{ mod } 0 \)
Error Checking

Yices “does no checking and can behave unpredictably if given bad input.”

To ensure soundness, the burden to produce correct input for the SMT solver is on our translation.
Experiments

Key experiences, based on “typical” proof obligations from the HOL4 library, and from work on machine-code verification:

- The SMT-LIB interface, due to its restrictions, does not add very much to existing proof procedures.
- Yices performs very well for proof obligations that involve bit-vector operations and linear arithmetic only.
- Yices’s support for quantifiers and λ-terms, however, could be improved.
Integration of Yices and SMT-LIB based solvers with HOL4

- SMT-LIB provides support for many solvers, but is restrictive.
- Yices has a rich native input language.
- Custom translations seem more worthwhile than sophisticated encodings into SMT-LIB format. (Unfortunate!)
- HOL4 available at http://hol.sourceforge.net/
Future Work

- Proof reconstruction (submitted; joint work with S. Böhme)
- A more expressive SMT-LIB format (Version 2.0 ?!)
- Considering context information (e.g., axioms and lemmas)
- Displaying models as counterexamples
Thank you!