SMT Solvers: New Oracles for the HOL Theorem Prover

Tjark Weber

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Workshop on Verified Software: Theories, Tools, and Experiments
Eindhoven, Netherlands

2 November 2009

Motivation

HOL4 is a popular interactive theorem prover.

Interactive theorem proving needs automation.

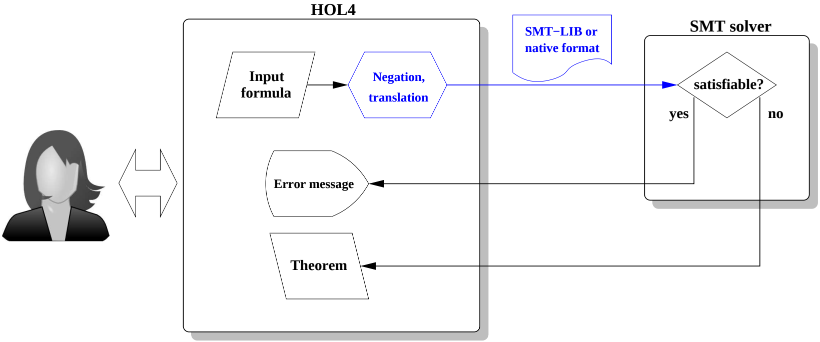
Motivation

HOL4 is a popular interactive theorem prover.

Interactive theorem proving needs automation.

$\Longrightarrow$ Use SMT solvers to decide SMT formulas.

System Overview



Higher-Order Logic

Polymorphic λ-calculus, based on Church's simple theory of types:

- $\sigma ::= \alpha \mid (\sigma_1, \ldots, \sigma_n)c$
- $t ::= x_\sigma \mid c_\sigma \mid (t_{\sigma\to\tau}\ t_\sigma)_\tau \mid (\lambda x_\sigma.\ t_\tau)_{\sigma\to\tau}$

Higher-Order Logic

Polymorphic λ-calculus, based on Church's simple theory of types:

- $\sigma ::= \alpha \mid (\sigma_1, \ldots, \sigma_n)c$
- $t ::= x_\sigma \mid c_\sigma \mid (t_{\sigma \to \tau}\ t_\sigma)_\tau \mid (\lambda x_\sigma.\ t_\tau)_{\sigma \to \tau}$

Extensive libraries:

- quantifiers (of arbitrary order)
- arithmetic (nat, int, real, ...)
- data types (tuples, records, bit vectors, ...)

$\Longrightarrow$ much of mathematics and computer science

Satisfiability Modulo Theories

Goal: To decide the satisfiability of (quantifier-free) first-order formulas with respect to combinations of (decidable) background theories.

$$\varphi ::= \mathcal{A} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

Satisfiability Modulo Theories: Example

Theories:

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

Problem: Is

$$x \leq y \wedge y \leq x + \mathrm{hd}\,(\mathrm{cons}\,0\,\mathrm{nil}) \wedge P\,(f\,x - f\,y) \wedge \neg\,P\,0$$

satisfiable in $\mathcal{I} \cup \mathcal{L} \cup \mathcal{E}$?

Translation from Higher-Order Logic

We must translate HOL formulas into the input language of SMT solvers.

1. SMT-LIB format
2. Yices's native format

SMT-LIB 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/

Yices's Native Format

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, λ-expressions
- `http://yices.csl.sri.com/`

Features: SMT-LIB vs. Yices

	SMT-LIB	Yices		SMT-LIB	Yices
int, real	✓	✓	let	(✓)	✓
nat, bool, $\rightarrow$		✓	λ-terms		✓
prop. logic	✓	✓	tuples		✓
equality	✓	✓	records		✓
FOL	✓	✓	data types		✓
HOL		✓	bit vectors	✓	✓
arithmetic	✓	✓			

Recursion & Abstraction

We translate HOL formulas by recursion over their term structure.

Abstraction is used to deal with unsupported terms/types.

Example: $P_{\alpha\to\mathrm{bool}}\ x_\alpha$

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, $\Longleftrightarrow$, $\Longrightarrow$, $\vee$, $\wedge$ 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.

Arithmetic (I)

SMT-LIB/Yices support directly:

- Types int, real, and (Yices only) nat
- Numerals (e.g., 3.14)
- Negation, addition, subtraction, multiplication
- Comparison operators $<$, $\leq$, $>$, $\geq$

Arithmetic (II)

For certain other HOL4 functions, e.g., min, max and abs, we introduce suitable definitions.

Example: abs $x_{\text{int}} \geq 0$

```
: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))
```

Let Expressions

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

Example: let $x = 1$ 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: $\forall f_{\alpha \to \beta}.\, \exists g_{\beta \to \alpha}.\, \forall x_{\alpha}.\, g\,(f\,x) = x$

```
(define-type a)
(define-type b)
(assert (not (forall (f::(-> a b))
    (exists (g::(-> b a))
        (forall (x::a) (= (g (f x)) x))))))
```

Anonymous and Higher-Order Functions

Yices provides a `lambda` construct, which is used to translate λ-abstractions. We first perform β-normalization and η-expansion in HOL4.

Functions of more than one argument are curried.

Function update $(a =+ b)\ f$ becomes `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

Monomorphisation

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 (α)foo and (β)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))
```

Data Types

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

Bit Vectors (II)

HOL4's w2w function is translated using either `bv-extract` or `bv-concat`, depending on the width of its argument and result.

Extracting a single bit from a bit vector, denoted by ′ in HOL4, is translated using Yices's `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.

Identifiers

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.

Semantic Differences

There are subtle semantic differences between certain HOL4 and (allegedly corresponding) SMT-LIB/Yices functions.

- Subtraction $m - n$ on naturals:

```
(define hol_num_minus::(-> nat nat nat)
    (lambda (x::nat y::nat)
        (ite (< x y) 0 (- x y))))
```

- $x \operatorname{div} 0$ and $x \bmod 0$

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.

Conclusions

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

Questions?

Thank you!