Solving Exists/Forall Problems With Yices

Bruno Dutertre (with Dejan Jovanović and Ian Mason)

SRI International

SMT Workshop 2015

Exists/Forall Problems

Example Synthesis Problem

- Find a function f such that $\forall y : \Phi(y, f(y))$
  (Φ specifies the properties we want for f)

Parameterization

- Look at a collection of functions f_x defined by a template parameterized by variables x.
  Example: linear functions $f_{a,b,c}(y_1, y_2) = a + by_1 + cy_2$
- The synthesis problem is now: find parameters x such that $\forall y : \Phi(y, f_x(y))$
  This is an exists/forall problem: check the satisfiability of

$$\exists x : \forall y : \Phi(y, f_x(y))$$

Example Application: Invariant Synthesis

Safety Property

- Given a state-transition system, we want to show that all reachable states satisfy a property $P(x)$
- We can try to find an inductive invariant $Q(x)$:

$$\forall x : I(x) \Rightarrow Q(x)$$
$$\forall x, x' : Q(x) \wedge T(x, x') \Rightarrow Q(x')$$
$$\forall x : Q(x) \Rightarrow P(x)$$

Template-Based Method

- Postulate that $Q(x)$ is of the form $F_{a,b,\ldots}$ for some unknown parameters $a, b, \ldots$
- Search for a solution to the following exists/forall problem:

$$\begin{aligned} \exists a, b, \ldots : \ \forall x, x' : \ & (I(x) \Rightarrow F_{a,b,\ldots}(x)) \\ & \wedge (F_{a,b,\ldots}(x) \wedge T(x, x') \Rightarrow F_{a,b,\ldots}(x')) \\ & \wedge (F_{a,b,\ldots}(x) \Rightarrow P(x)) \end{aligned}$$

More Examples

Template-based Synthesis

- loop-free programs (Jha et al., 2011, Jha et al. 2010)
- switching logic for hybrid systems (Taly et al., 2011)
- controller synthesis (Cheng et al., 2013, Sturm and Tiwari, 2013)

Template-based Verification

- Lyapunov Functions: to show stability of dynamical systems
- Barrier Certificates for Hybrid Systems: (Prajna, 2003, . . .)

How to Solve It?

$$\exists x : \forall y : \Phi(x, y)$$

Quantifier-Elimination Methods

- Rewrite $\forall y : \Phi(x, y)$ into an equivalent quantifier-free formula $\Phi'(x)$
- Search for x that satisfies $\Phi'(x)$

Two-Solver Approach

- E-solver: search for candidates x
- F-solver: given a candidate x_0, try to show it's not good:
  search for y such that $\neg\Phi(x_0, y)$
- Repeat until we find a good candidate or we have exhausted all candidates

Comparison/Tradeoffs

Quantifier Elimination

- Applicable to real arithmetic (linear and non-linear), Boolean problems, etc.
- Does more than we need: $\Phi'(x)$ characterizes all solutions, we just need one
- Typically very expensive (huge blowup in formula size)
- Requires specialized tools (e.g., CAD algorithm)

Two Solvers

- Existing SMT or SAT solvers can be used
- Other approaches are applicable: random sampling, numerical methods
- Potentially more scalable than quantifier elimination (no immediate blow up)
- Issues
  - How to efficiently combine the two solvers?
  - How to guarantee termination?

EF-Solver Algorithm

$i := 0$

$C_0(x) :=$ initial constraints on x

repeat

  find x_i that satisfies $C_i(x)$ [E-Solver]

    if no x_i is found, return **unsat**

  search for y_i that satisfies $\neg\Phi(x_i, y)$ [F-Solver]

    if no y_i is found, then x_i is a solution;

      return **sat**

  generalize from y_i: compute a constraint $G(x)$ such that

    1) $G(x_i)$ is true

    2) $G(x) \Rightarrow (\exists y : \neg\Phi(x, y))$

  $C_{i+1}(x) := C_i(x) \wedge \neg G(x)$

  $i := i + 1$

end

Key Procedure: Generalization

Three Methods Implemented in Yices

- baseline: just remove x_i: $G(x) := (x = x_i)$
- generalize by substitution: $G(x) := \neg\Phi(x, y_i)$
- better: local quantifier elimination
  - find an implicant $J(x, y)$ for $\neg\Phi(x, y)$ using x_i and y_i:
    - $J(x, y)$ is a conjunction of literals
    - $J(x, y) \Rightarrow \neg\Phi(x, y)$ holds
    - $J(x_i, y_i)$ is true
  - construct $G(x)$ by eliminating the y variables from $J(x, y)$

Convergence

Termination Guarantees

- obvious if the x variables have a finite domain
- otherwise, termination depends on the generalization procedure
  - if the y variables have a finite domain, then generalization by substitution ensures termination
  - for infinite domains: some form of quantifier elimination is required

Example: in linear arithmetic

$$\exists x \in \mathbb{R} : \forall y \in \mathbb{R} : x < y$$

This is unsat but EF-solving using generalization by substitution doesn't converge.

Implicant Construction

Goal

- given a formula Φ and a model $\mathcal{M}$ of Φ, construct a conjunction of literals I such that $I \Rightarrow \Phi$ and $\mathcal{M} \models I$

Procedure

- Top-down traversal of Φ, using $\mathcal{M}$ to guide the search.
- This relies on the fact that we can evaluate formulas in $\mathcal{M}$.
- Example: to find an implicant for $(\phi_1 \vee \ldots \vee \phi_n)$,
  - search for ϕ_i that's true in $\mathcal{M}$
  - then recursively compute an implicant of ϕ

(Simplified) Implicant Construction

$$
\begin{array}{rcll}
Imp^+(l) & := & l & \\
Imp^+(f_1 \vee f_2) & := & Imp^+(f_1) & \text{if } f_1 \text{ is true in } \mathcal{M} \\
 & := & Imp^+(f_2) & \text{otherwise} \\
Imp^+(f_1 \wedge f_2) & := & Imp^+(f_1) \wedge Imp^+(f_2) & \\
Imp^+(\neg f) & := & Imp^-(f) & \\
 & & & \\
Imp^-(t = 0) & := & (t > 0) & \text{if } t \text{ has a positive value in } \mathcal{M} \\
 & := & \neg(t \geqslant 0) & \text{if } t \text{ has a negative value in } \mathcal{M} \\
Imp^-(t > 0) & := & \neg(t > 0) & \\
Imp^-(t \geqslant 0) & := & \neg(t \geqslant 0) & \\
Imp^-(f_1 \vee f_2) & := & Imp^-(f_1) \wedge Imp^-(f_2) & \\
Imp^-(f_1 \wedge f_2) & := & Imp^-(f_1) & \text{if } f_1 \text{ is false in } \mathcal{M} \\
 & := & Imp^-(f_2) & \text{otherwise} \\
Imp^-(\neg f) & := & Imp^+(f) &
\end{array}
$$

Real Implicant Construction

If-then-else

- Implicant for $1 + (\texttt{ite}\ c\ x\ y) \geqslant 0$ can be either $c \wedge 1 + x \geqslant 0$ or $\neg c \wedge 1 + y \geqslant 0$.

Distinct atoms

- $(\texttt{distinct}\ t_1\ \ldots\ t_n)$ is converted to a conjunction of inequalities if $t_1, \ldots, t_n$ are arithmetic terms.
- This is done by sorting $t_1, \ldots, t_n$ according to their values in the model.

Boolean terms

- In some contexts, we treat Booleans as terms, in other contexts we treat them as formulas.
- Example:
  - $(x = u)$ is treated as an atom if x is a Boolean variable
  - $(t = u)$ is treated as $(t\ \wedge\ u)\ \vee\ (\neg t \wedge \neg u)$ if t and u are not variables

Variable Elimination

Goal

- We have an implicant $J(x, y)$ that is true in a model $\mathcal{M}$
- We want to eliminate the variables y from $J(x, y)$
- We could try to construct a $G(x)$ that's equivalent to $\exists y : J(x, y)$
- In our context, it is enough to obtain an under-approximation:

$$G(x) \Rightarrow \exists y : J(x, y)$$

such that $\mathcal{M} \models G(x)$

For linear (and non-linear) arithmetic, we can do this efficiently using Model-Guided Virtual Term Substitution

Virtual Term Substitution for Linear Arithmetic

Weispfenning, 1988, Loos & Weispfenning, 1993

- To eliminate y from a linear arithmetic formula $\exists y : \phi(x, y)$, construct an elimination set for y in $\phi(x, y)$
- An elimination set is a finite set T of terms that do not contain y and such that

$$(\exists y : \phi(x, y)) \Leftrightarrow \bigvee_{t \in T} \phi(x, t)$$

- T can be constructed syntactically from the atoms of ϕ

Example

- For $(\exists y : 3x + 1 < y \wedge y < x + 2)$, Weispfenning's procedure gives

$$T = \left\{ 3x, 3x + 1, 3x + 2, x + 1, x + 2, x + 3, \frac{(3x + 1) + (x + 2)}{2} \right\}$$

Model-Guided Virtual Term Substitution

Idea

- We start from an elimination set T such that

$$(\exists y : \phi(x, y)) \Leftrightarrow \bigvee_{t \in T} \phi(x, t)$$

- Since we can under-approximate, it's enough for us to pick a single term t_0 in T

$$\phi(x, t_0) \Rightarrow (\exists y : \phi(x, y))$$

- We also have a model $\mathcal{M}$ of $\phi(x, y)$ so we use $\mathcal{M}$ to find a suitable t_0

Example

$$\exists y : 3x + 1 < y \wedge y < x + 2$$

$$T = \left\{ 3x, 3x + 1, 3x + 2, x + 1, x + 2, x + 3, \frac{(3x + 1) + (x + 2)}{2} \right\}$$

Model: $x \mapsto 0$ and $y \mapsto 1.5$

- We pick

$$t_0 = \frac{(3x + 1) + (x + 2)}{2}$$

then $\phi(x, t_0)$ reduces to $x < 1/2$

Variable Elimination as Implemented in Yices

Input

- The implicant construction produces a conjuction of arithmeitc inequalities and equalities

Hybrid Approach

- eliminate variables that occur in equalities (Gaussian elimination)
- use Fourier-Motzkin if it's cheap
- use virtual-term subsitution as a last step.

Other Tricks

Preprocessing

- rewrite the problem to the following form:

$$\begin{aligned} & \exists x : A(x) \\ & \quad \wedge \quad (\forall y_1 : B_1(y_1) \;\Rightarrow\; \phi_1(x, y_1)) \\ & \qquad\qquad \vdots \\ & \quad \wedge \quad (\forall y_n : B_n(y_n) \;\Rightarrow\; \phi_n(x, y_n)) \end{aligned}$$

- this tends to give smaller problem instances to the F-solver
- this helps learning the initial constraints on x:
  (i.e., we search for y_i that satisfies $B_i(y_i)$)

Sampling Approach

- to find "diverse" y_i, we use a bounded variant of all-SAT

Implementation Status

EF Solver

- Part of Yices since version 2.3.
- Available at `http://yices.csl.sri.com/`
  - The EF solver supports linear real arithmetic, bitvector, and Boolean constraints
  - The input must be given in the Yices language
  - Generalization and implicant construction are in the API

Example Input

```
(define x::real)
(define y::real)

(assert (/= x y))
(assert (forall (z::real) (=> (> y z) (> x z))))
(ef-solve)
(show-model)
```

Performance

Benchmarks

- 9 problems from Cheng et al. 2013, all relatively small:
  - 4 examples related to control (encoded using bitvectors)
  - 5 priority synthesis for distributed systems (pure Boolean)

Run time (ms)

Solver	Control				Prio. Synth.				
Cheng	0	0	10	10	20	20	170	2430	6740
Yices	0	0	4	8	4	4	16	96	176

Iterations

Solver	Control				Prio. Synth.				
Cheng	2	2	3	7	4	4	18	111	11
Yices	1	1	4	7	1	1	10	68	6

Hardware Application (Gascón, et al., 2014)

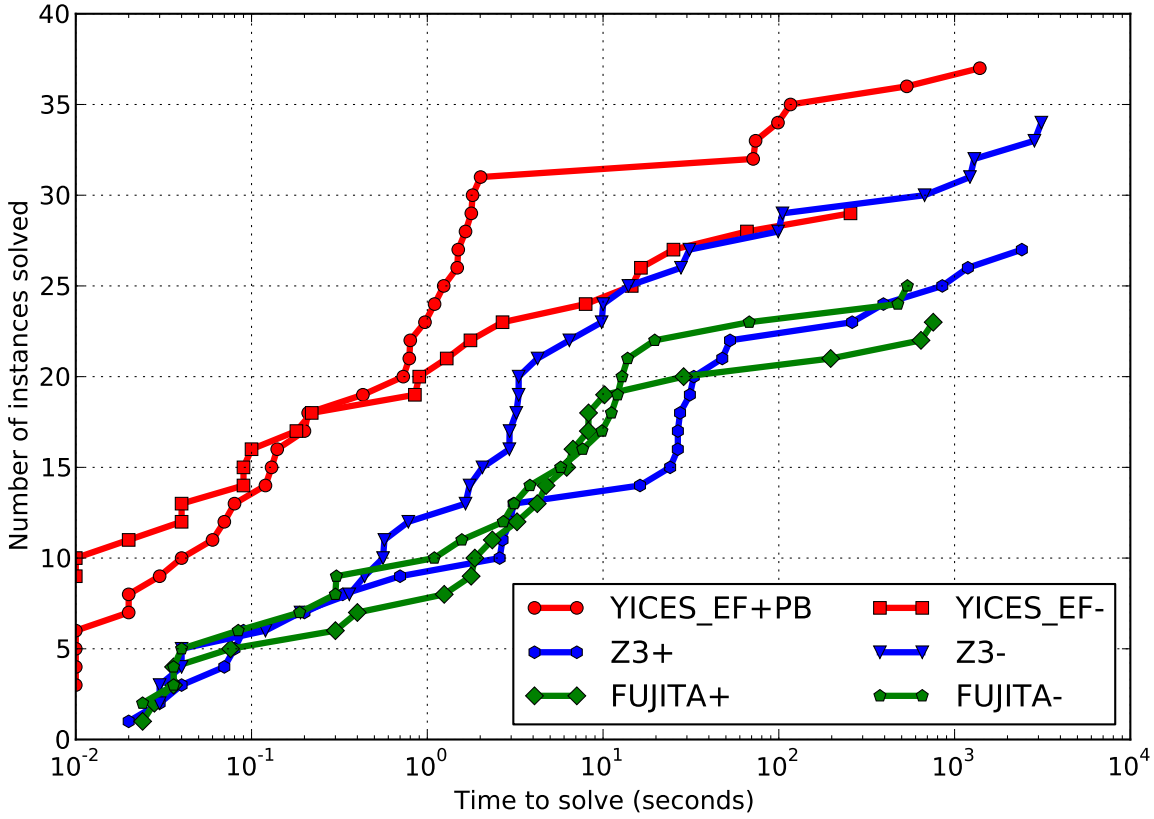
Reverse Engineering of Hardware

- Check whether a low-level circuit description (netlist) is an instance of a given circuit pattern
- Formulated as an exists/forall problem over bitvectors
- Tested with Yices, Z3, several QBF solvers, and an EF-procedure for 2QBF (Janota and Silva, 2011)

Results

- Yices and Z3 work best on these benchmarks
- Generic QBF solvers do not work well at all

Empirical Results



Program Sketching: Synudic (Tiwari & Gascón)

Synudic

- Program synthesis framework
- Key idea: dual interpretation
  - Basic operations are defined by a signature, an operational, and a *non-operational* semantics
  - The non-operational semantics gives constraints on how operators can be composed
- Uses EF-solving with Yices as a backend
- More details at CADE'2015

  `http://www.csl.sri.com/users/tiwari/softwares/auto-crypto`

Summary

Exists/Forall Solving

- Easy and useful extension of quantifier-free SMT solving
- The key is to generalize from models
- Given y_i and x_i such that $\neg\Phi(x_i, y_i)$, we want to remove x_i and as many other x-candidates as we can from consideration
- This amounts to computing a formula $G(x)$ such that
  - $G(x_i)$ is true
  - $G(x) \Rightarrow (\exists y : \neg\Phi(x, y))$

For Linear Arithmetic

- We can compute a G efficiently by
  - computing an implicant J from the model
  - applying quantifier elimination to J based on virtual term substitution, guided by the model

Coming Soon

SMT-LIB Frontend

- we're working on a front-end for LRA and BV that uses standard SMT-LIB 2 syntax

More Logics

- The next Yices release will include MCSAT and support for non-linear arithmetic (QF_NRA)
- We'll extend EF-solver to non-linear real arithmetic after that
- We're also planning to support linear integer arithemtic (Cooper's algorithm).