Synthesis Using EF-SMT Solvers

Ashish Tiwari
SRI International

Collaborators

Bruno Dutertre (SRI)
Sumit Gulwani (MSR)
Vijay Anand Korthikanti (UIUC)
Jan Leike (Australia)
Sharad Malik (Princeton)
Thomas Sturm (Munich)
Ankur Taly (Google)

Adrià Gascón (SRI)
Dejan Jovanovic (SRI)
Susmit Jha (Intel)

Sanjit Seshia (UC Berkeley)
Pramod Subramanyan (Princeton)
Ramarathnam Venkatesan (MSR)

EF-SMT or $\exists\forall$-SMT

$$\exists \vec{a} : \forall \vec{x} : \phi(\vec{a}, \vec{x})$$

Important factor: theory of ϕ

Many synthesis problems can be transformed into $\exists\forall\phi$ problems

Sat/SMT is to verification as QBF/EF-SMT is to synthesis

Synthesis

Verification: Given M, ϕ, determine if $M \models \phi$

Synthesis: Given partial model $M(?)$ and ϕ, determine if there exists a completion M of $M(?)$ s.t. $M \models \phi$

Value: Applications where completing $M(?)$ is not intuitive/easy

Philosophy: Lift formal verification techniques to synthesis

Synthesis as $\exists\forall\phi$: Version 1

Take inspiration from bounded model checking:

$$M(\vec{c}, \vec{u}) \models \phi$$
$$\Uparrow$$
$$M(\vec{c}, \vec{u}) \models \phi^k, \qquad \text{for some } k$$
$$\Uparrow$$
$$\exists \vec{c} : \forall \vec{x_0}, \vec{x_1}, \ldots, \vec{x_k} : \exists \vec{u_0}, \ldots, \vec{u_k} : \text{ quantifier-free FO formula}$$
$$\Uparrow$$
$$\exists \vec{a}, \vec{c} : \forall \vec{x} : \exists \vec{u} : \text{ quantifier-free FO formula}$$

Synthesis as $\exists\forall\phi$: Version 2

Take inspiration from induction- and abstraction-based verification:

$$M(\vec{c},\vec{u}) \models \phi$$
$$\Uparrow$$
$$\exists\Phi,\vec{c},\vec{u} : ((M(\vec{c},\vec{u}) \models \Phi) \ \wedge \ (\Phi \Rightarrow \phi))$$
$$\Uparrow$$
$$\exists\Phi,\vec{c} : \forall\vec{x} : \exists\vec{u} : \ \text{quantifier-free FO formula}$$
$$\Uparrow$$
$$\exists\vec{a},\vec{c} : \forall\vec{x} : \exists\vec{u} : \ \text{quantifier-free FO formula}$$

The last step is performed by choosing a template for Φ

Verification :: Synthesis

Verification : $\exists\forall$
does there exist a certificate that proves the property for the system?

Synthesis: $\exists\forall\exists$
synthesis variables that are independent of the state go inside the first $\exists$
synthesis variables that are dependent on the state go inside the second $\exists$

Usually link verification to one and synthesis to two quantifiers

Note: Not just parameter synthesis, the synthesis variables could encode how subsystems are composed...

Synthesis Challenge

Automated synthesis is difficult

- Search space for models is very large
  domain of first $\exists$ quantifier large
- For each potential model, we have to solve a verification problem
  inner $\forall$ problem difficult

Overcoming the Synthesis Challenge

- Reduce outer $\exists$ complexity:
  - Use templates to reduce synthesis search space
  - Bound all synthesis variables
- Reduce inner $\forall$ complexity:
  - consider domains where verification is easy (for e.g., straight-line programs)
  - Replace "verification" check by a weaker check (as in BMC)
  - Search for small (strong) verification proofs

Syn1: Straight-line Program Synthesis

Problem: Given library functions f_i's, and a desired f, find a way to compose f_i's to generate f.

$$\exists P : \forall x : f(x) = P(x), \quad P \text{ a straight-line program composing } f_i\text{'s}$$
$$\Downarrow$$
$$\exists \pi : \forall x : F_{\text{spec}}(x) = f_{\pi(1)}(f_{\pi(2)}(f_{\pi(3)}(x)))$$

Reduce synthesis search space:

- fix length of program
- fix upper bound on number of each f_i

Verification problem: Straight-line program equivalence

Syn1 Example

Specification: Turn-off rightmost contiguous 1 bits

Example: 010101100 $\mapsto$ 010100000

Budget: two addition, at most four bitwise Boolean operators, constant 1, -1

Synthesized Program:

1. $o_1 := x + (-1)$;
2. $o_2 := o_1 | x$;
3. $o_3 := o_2 + 1$;
4. return $o_3 \& x$;

Correctness on sample input:

$$010101100 \mapsto 010101011 \mapsto 010101111 \mapsto 010110000 \mapsto 010100000$$

Other examples: Bitvector tricks from Hacker's delight

Synthesis: $\exists\forall$ bit-vector theory

Syn1.1: Loop-free Program Synthesis

Problem: Given library functions f_i's including an if-then-else, and a desired f, find a way to compose f_i's to generate f.

Step to feasibility: As in Syn1.

Syn1.1 Example

Specification: Obfuscated code

Example: We are given

```
if (h(x))
   if (x*(x+1)% 2 == 1) y := f(x) else y := g(x)
else y := f(g(x))
```

Components Budget: f, g, h, if-then-else

Synthesized Program:

```
o := g(x);
if (h(x)) y := o; else y := f(o);
```

Correctness: Equivalence of the two loop-free programs

Syn2: Loop Synthesis

Problem: Given incomplete loop body and desired post, find a way to complete the loop body so that correctness holds.

procedure product(x_0, y_0):
    $s := 0$;
    $y := y_0$;
    while $(y \neq 0)$:
        $s := c_1 x_0 + c_2 y_0 + c_3 y + c_4 s + c_5$;
        $y := y - 1$;
    return s; $[s = x_0 * y_0]$

Syn2: Synthesize Code and Correctness

Let loop invariant be an arbitrary degree 2 polynomial

$$a_0 x_0^2 + a_1 y_0^2 + a_2 y^2 + a_3 s^2 + a_4 x_0 y_0 + \ldots + a_{14}$$

Now the synthesis formula is:

$\exists c_1, \ldots, c_5 :$
$\exists a_0, \ldots, a_1 4 :$
$\forall x_0, y_0, s, y :$
    loop-invariant is indeed a loop invariant
    loop-invariant and negation of condition implies post

$\exists\forall$ theory of reals

Syn3: Fault-tolerant Distributed Algorithms

A set of m (communicating) machines arranged in some topology want to satisfy a given temporal property. Given the state space of the machines, find the state update function for the m machines.

Do so assuming some f are faulty

Fault assumptions can be malicious, asymmetric, transient.

Both the synthesis search space, and the verification problem need to be simplified

Syn3 Example

Each machine has a private Boolean value v_i

Each machine has a vector of Boolean values – its knowledge about the private value of all the other machines – consistency vector

Interactive consistency: n processes, out of which at most f can be faulty, satisfy interactive consistency if the consistency vectors of some $n - f$ processes are identical

Desired property: FG(interactive consistency)

Each machiine updates its v_i by $f_i(v'_1, \ldots, v'_m)$, where v'_j is the value it receives from j-th machine (may not equal v_j if j is faulty)

$$\exists f_i : S_1 || \ldots || S_m \models \mathrm{FG}(\text{interactive-consistency})$$

Syn3 Example: Simplifying Verification

Typically also want algorithms to be self-stabilizing

I.e., initial state can be arbitrary

In this case, $\mathsf{FG}\phi$ is implied by, say, $\mathsf{XXXX}\phi$

Thus, for a fixed m, we get an 2-QBF formula

We can synthesize Pease, Shostak, and Lamport's algorithm for interactive consistency that works in presence of permanent, malicious and asymmetric (Byzantine) faults

Other examples: Dijkstra's self-stabilizing mutual exclusion algorithm, interactive consistency for transient Byzantine faults

Syn4: Circuit Understanding

Goal: Discover high-level description of a given combinational circuit (verilog netlist)

Permutation Independent Equivalence Checking: Is circuit equivalent to, say an adder, for some permutation of the input and output signals

Permutation Independent Conditional Equivalence Checking: Is circuit equivalent to, say an adder, for some permutation of the input and output signals and some setting of the control input signals

Syn4: Simplifications

Real challenge for large netlists

Reducing the synthesis search space

- templates – user can specify subsets of input signals whose permutation may be one input word; same for control and output signals
- signatures – necessary conditions on the synthesis variables obtained from analyzing the target function
  some over-approximation of QE of $\forall\phi$
  type constraints that eliminate solutions (that will not work)

Syn4: Examples

Flattened netlists generated using Synopsys Design Compiler

From high-level behavioral verilog collected from ISCAS'85 benchmarks, ALU from a academic processor implementation, etc.

Template library contains adders, subtracters, shifters, multipliers, counters of varying bitwidths

Equal number of positive and negative synthesis instances used

Successfully solved 37/40 instances with QBF preprocessor Bloqqer and EF-Yices

Syn5: Security Schemes

Goal: Automatically synthesize encryption schemes, message exchange protocols, etc.

Similar to straight-line program synthesis

Typical protocols are 10-20 line programs

Requirement: Meet some security level

A Template Language

A class of straight-line programs with $na + nb + 4$ lines:

```
(library
  (G 1) (H 1) (oplusr 2) (oplus 2) (identity 1))
(blocks
   (lm 1 ((input m::(bool-to-bv false false false false true))))
   (lr 1 ((input r::(bool-to-bv false false false true false))))
   (l1 na ( (oplusr (lm -) (lr -)) (G (lr -)) (H (lm -))))
   (l2 2 ( (identity (l1) ) ))
   (l3 nb ( (oplus (l2 -) (l2)) (H (l2)) (G (-)) )))
```

Template Language Features

- broad in its range: from writing concrete program to arbitrary program of length l
- hence, can restrict the synthesis search space
- length of program (blocks) can be left parametric
- Each variable is associated to two entities – its value and its type
- Can include type restrictions to further restrict synthesis search space, and also describe properties
- Is internally compiled into EF-Yices: can use arbitrary yices expressions too

Syn5: Example

Synthesizing OEAP encryption scheme

Using the part template shown above

Interpret messages over bitvectors, and all operations mapped to bitvector operations

Define a type that conservatively tracks which terms are essentially random

Requirement: Output terms are 'random' and there is an efficient decryption algorithm

Syn5: Examples of Synthesized Schemes

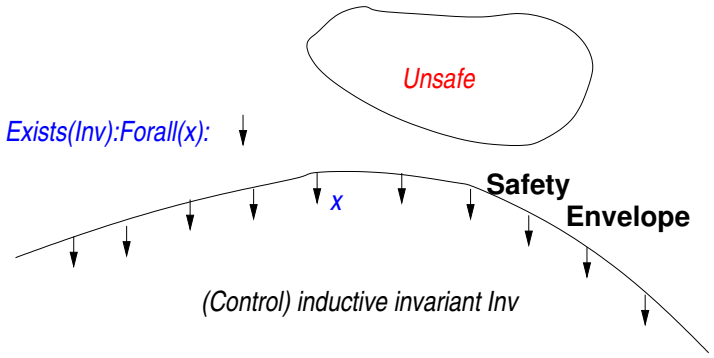
- $f(G(r)||m + G(r))$
- $f(r||m + G(r))$
- $f(r + H(m)||m + G(r + H(m)))$
- $f(G(r + H(m))||r + G(r + H(m)))$
- $f(m + G(r)||H(m + G(r)) + r)$

Why?

- padding with zero not modeled
- and the resulting security issues

Syn6: Controller Synthesis for Safety

Φ = (Global) Control inductive invariant



$\exists$(control inductive invariant that implies safety)

$\exists(set)$: (set is a control inductive invariant) and (set implies safety)

$\exists(V(\vec{x}) \geq 0) : \forall \vec{x} : \exists u : (V(\vec{x}) = 0 \Rightarrow V(F(\vec{x}, u, 0^+)) \geq 0)$

$\wedge(V(\vec{x}) \geq 0 \Rightarrow \text{Safe}(\vec{x}))$

Syn6: Controller Synthesis for Safety

Two things:

- Safe controller will be a fallback controller, so as well make it "bang-bang":

$$\exists \vec{a}, \vec{c} : \forall \vec{x} : \exists \vec{u} : \phi$$
$$\Uparrow$$
$$\exists \vec{a}, \vec{c} : \forall \vec{x} : (\phi(u_{min}) \vee \phi(u_{max}))$$

- Rewrite $V(F(\vec{x}, u, 0^+)) \geq 0$ to eliminate F

Syn6: Controller Synthesis for X

Can be similarly converted to an $\exists\forall\exists$ and if domain of u is finite, then to an $\exists\forall$ problem

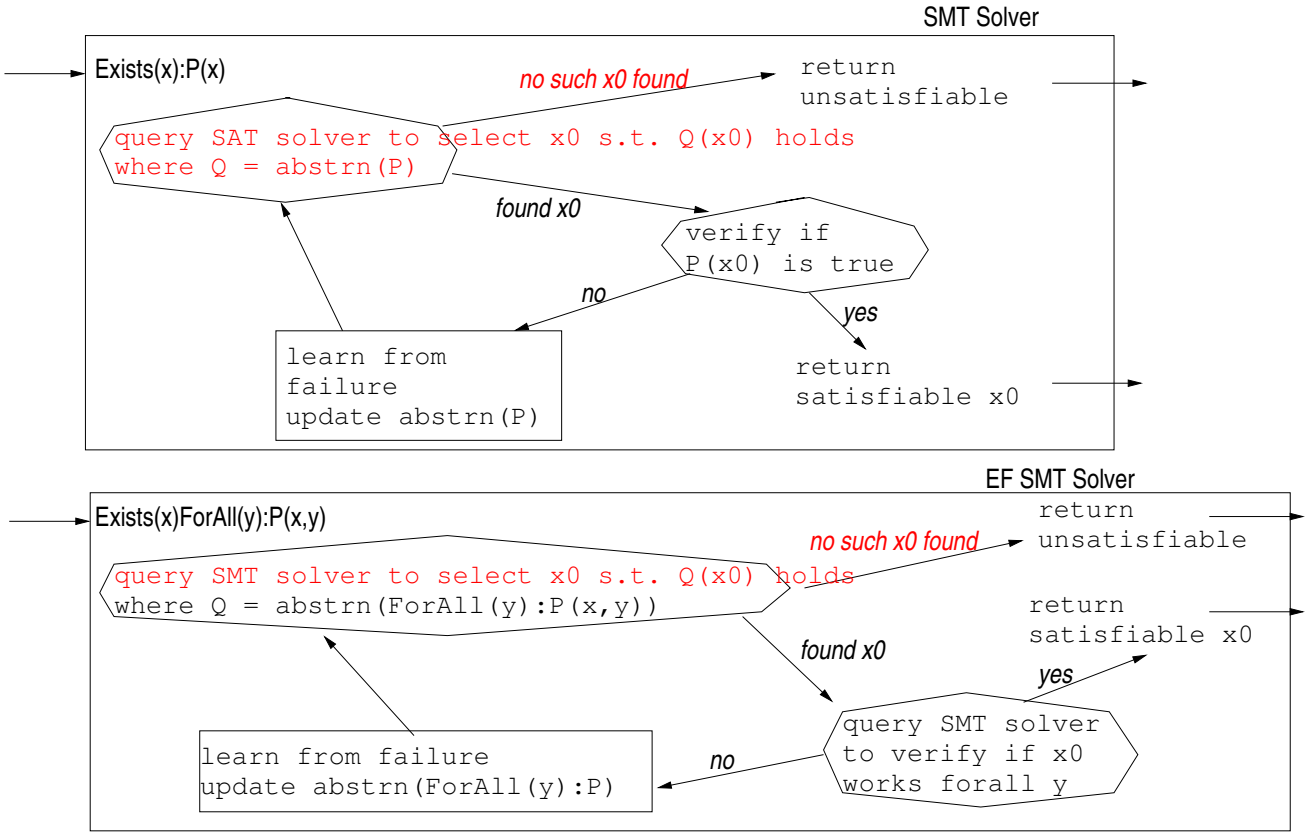
Applications:

- switching logic synthesis
- adaptive cruise controller

How to Solve $\exists\forall$ Problems?

EF Solving

Constraint Solving Using Abstraction Refinement
Guided by Learning via Counter Examples



EF Solving: Learning

$$\exists x : E(x) \wedge \forall y : B(x, y) \Rightarrow C(x, y)$$

Find a s.t. $E(a)$; if none, then return Unsat

Find b s.t. $B(a, b) \wedge \neg C(a, b)$; if none, return Sat

Update $E := E \cup \{B(x, b) \Rightarrow C(x, b)\}$

Convergence can be slow

perform approximate quantifier elimination on $\forall y : B(x, y) \Rightarrow C(x, y)$ to get constraints on x

E.g., equality resolution

Terminates for finite domains

EF Solving: Arithmetic Domains

Several different approaches:

- dedicated QE procedures
- Use duality theorems to replace $\forall$ by $\exists$
- Some combination of simplification, QE, simulation,

EF Solving: Simulation Guided Synthesis

$$\exists \vec{a} : \forall \vec{x} : (V \geq 0 \Rightarrow gap \geq 0) \ \wedge \ (V = 0 \Rightarrow dV/dt > 0)$$

1. $S :=$ Sample the reachable state space
   - simulation/trace data
   - constraint solving on guard for $\forall$
2. Solve $\exists \vec{a} : \bigwedge_{\vec{x} \in S} : V \geq 0$
   - Find a suitably large safety envelope that contains all points
   - Using the SMT solver Yices – note linear
3. Shrink the safety envelope minimalistically to guarantee safety
   - All but one $\exists$ variables is fixed
   - Constant term d is the only $\exists$ variable remaining

- Use quantifier elimination

EF Solving: NRA Fragment

ϕ: Nonlinear real arithmetic

$$\exists \vec{a} : \forall \vec{x} : \phi(\vec{a}, \vec{x})$$
$$\Uparrow$$
$$\exists \vec{a}, \vec{p} : \forall \vec{x} : \bigwedge_i q_i(\vec{p}, \vec{a}, \vec{x}) = 0$$
$$\Uparrow$$
$$\exists \vec{a}, \vec{b} : \forall \vec{x} : \bigwedge_i r_i(\vec{a}, \vec{b}, \vec{x}) = 0$$
$$\Uparrow$$
$$\exists \vec{a}, \vec{b} : \bigwedge_i s_i(\vec{a}, \vec{b}) = 0$$

So problem reduces to $\exists \vec{a} : \phi(\vec{a})$

EF-NRA $\mapsto$ E-NRA: Example

$$\exists a, b, u, v, w \forall s, x_0, y_0, y, s_1, y_1 :$$

$$s = ax_0y_0 + bx_0y \wedge s_1 = s + x_0 \wedge y_1 = y - 1 \Rightarrow s_1 = ax_0y_0 + bx_0y_1$$

$$\Uparrow$$

$$s_1 - ax_0y_0 - bx_0y_1 = u(s - ax_0y_0 - bx_0y) + v(s_1 - s - x_0) + wx_0(y_1 - y + 1)$$

$$\Uparrow$$

$$1 = v \wedge -a = -ua \wedge -b = w \wedge 0 = u - v \wedge 0 = -ub - w \wedge 0 = -v + w$$

Other Approaches for EF Solving

From "input-output examples":

Finite synthesis:

$$\exists \forall \phi$$
$$\exists \bigwedge_i \phi$$

Distinguishing input technique:

$$\exists \forall \phi$$
$$\exists c \exists c' : \bigwedge_i \phi(c) \wedge \phi(c') \wedge c \neq c'$$

Conclusion

- convert synthesis problem to $\exists\forall$ problem

EF-SMT:

- integrate various different approaches
- into an effective strategy for EF-SMT solving