Syntax-Guided Synthesis in SMT: A View from Inside the Solver

Andrew Reynolds
March 1, 2021
Satisfiability Modulo Theories (SMT) Solvers

• Fully automated reasoners with many applications
  • Verification, *Synthesis*, Symbolic Execution, Theorem Proving, Security Analysis

• SMT solver CVC4
  • Open source, available at: [https://cvc4.github.io/](https://cvc4.github.io/)

• Acknowledgements:
  • Cesare Tinelli, Clark Barrett, Haniel Barbosa, Andres Noetzli, Aina Niemetz, Mathias Preiner
  • Rest of CVC4 development team (past and present)
  • Viktor Kuncak
There exists a function $f$ for which property $P$ holds for all $x$. 

$$\exists f. \forall x. P(f, x)$$
Synthesis Conjectures *Modulo T*

$$\exists f . \forall x . P(f, x)$$

There exists a function $f$ for which property $P$ holds for all $x$.

Property $P$ is in **background theory** $T$, e.g. linear arithmetic

$$\exists f . \forall x . f(x+1) \geq f(x)$$

$\Rightarrow$ Satisfiability Modulo Theories (SMT)
There exists a function $f$ for which property $P$ holds for all $x$.
There exists a function $f$ for which property $P$ holds for all $x$.

The body of $f$ is generated by the above grammar with start symbol $A$.

$\Rightarrow Syntax-guided synthesis "\text{SyGuS}"$ [Alur et al 2013]
Enumerative Cex-Guided Inductive Synthesis (CEGIS)

syntax(f):
A→A+A|¬A|x|y|0|1|ite(B,A,A)
B→B∧B|¬B|A=A|A≥A|⊥

spec(f):
∀xy. f(x,y) = f(y,x) + f(0,1)

-Solution Enumerator

-Solution Verifier

Enumerative Cex-Guided Inductive Synthesis (CEGIS)

**Syntax**

\[
\text{syntax}(f): \\
A \rightarrow A + A | -A | x | y | 0 | 1 | \text{ite}(B, A, A) \\
B \rightarrow B \land B | \neg B | A = A | A \geq A | \bot
\]

**Specification**

\[
\text{spec}(f): \\
\forall xy. f(x, y) = f(y, x) + f(0, 1)
\]

**Solution**

\[
f := \lambda xy. x?
\]
Enumerative Cex-Guided Inductive Synthesis (CEGIS)

**syntax(f):**

\[
\begin{align*}
A & \rightarrow A + A \\
\neg A & \leftarrow x | y | 0 | 1 | \text{ite}(B, A, A) \\
B & \rightarrow B \land B | \neg B | A = A | A \geq A | \perp
\end{align*}
\]

**spec(f):**

\[
\forall xy. f(x, y) = f(y, x) + f(0, 1)
\]

---

**Solution Enumerator**

\[ f := \lambda xy. 0 \]?

\[ f(2,1) = f(1,2) + f(0,1) \]

\[ f := \lambda xy. y \]?

\[ f(0,1) = f(1,0) + f(0,1) \]

\[ f := \lambda xy. x \]?

\[ f := \lambda xy. 0 \]

---

**Solution Verifier**

\[ f := \lambda xy. 0 \]

...new candidate \( \lambda xy. 0 \) has no counterexamples wrt \( \text{spec}(f) \)
Enumerative Cex-Guided Inductive Synthesis (CEGIS)

**Syntax**

\[
\begin{align*}
syntax(f): \\
A & \rightarrow A + A \\ 
- A & \rightarrow x | y | 0 | 1 | \text{ite}(B, A, A) \\ 
B & \rightarrow B \land B | \neg B | A = A | A \geq A | \bot
\end{align*}
\]

**Specification**

\[
\begin{align*}
spec(f): \\
\forall x, y. f(x, y) &= f(y, x) + f(0, 1)
\end{align*}
\]

**Solution Enumerator**

\[
f := \lambda xy. \text{?}
\]

**Solution Verifier**

\[
f := \lambda xy. \text{?}
\]

\[
f := \lambda xy. 0
\]

⇒ Terms \(x, y, 0, \ldots\) are a (fair) enumeration of terms generated by \(syntax(f)\)
CEGIS using SMT solvers

**spec(f):**
\[ \forall x y. f(x, y) = f(y, x) + f(0, 1) \]

**syntax(f):**
\[
\begin{align*}
A & \rightarrow A + A | -A | x | y | 0 | 1 | \text{ite}(B, A, A) \\
B & \rightarrow B \land B | \neg B | A = A | A \geq A | \bot
\end{align*}
\]
CEGIS inside an SMT solver

**Solution Enumerator**

**Solution Verifier**

Syntax ($f$):

\[ A \rightarrow A + A | -A | x | y | 0 | 1 | \text{ite}(B, A, A) \]

\[ B \rightarrow B \land B | \neg B | A = A | A \geq A | \perp \]

Spec ($f$):

\[ \forall xy. f(x, y) = f(y, x) + f(0, 1) \]

SMT Solver (CVC4)

[Reynolds et al CAV 2015]

- ✔ Synthesis algorithms that use internal state of SMT solver
- ✔ Tight integration between enumerator and verifier
In This Talk

- Synthesis approaches used by SMT+SyGuS solver CVC4:
  1. Counterexample-guided quantifier instantiation
  2. Smart Enumerative SyGuS
  3. Fast Enumerative SyGuS

- Internal applications of SyGuS for SMT solvers
- Future work
Synthesis Solver CVC4

\[ \exists f. \forall x. P(f, x) \]

CVC4

\[ f = \lambda x. t(x) \]
Synthesis Solvers in CVC4

\[ \exists f. \forall x. P(f, x) \]

CVC4

Counterexample Guided QI

Smart Enumerative

Fast Enumerative

[Reynolds et al CAV 2015, FMSD 2017, Niemetz et al CAV 2018]

[Reynolds et al CAV 2015, IJCAR 2018]

[Reynolds et al CAV 2019]

⇒ Best approach to apply depends on the conjecture
Approach #1:
Counterexample-Guided Instantiation
Synthesis via Counterexample-Guided Instantiation

• Some synthesis conjectures are *essentially first-order*:

\[ \neg \exists f. \forall x y. f(x, y) \geq x \land f(x, y) \geq y \land (f(x, y) = x \lor f(x, y) = y) \]

“\( f(x, y) \) is the maximum of \( x \) and \( y \)”
Synthesis via Counterexample-Guided Instantiation

\[
\neg \exists f. \forall xy. f(x, y) \geq x \land f(x, y) \geq y \land (f(x, y) = x \lor f(x, y) = y)
\]

Anti-skolemize

\[
\neg \forall xy. \exists z. \ z \geq x \land z \geq y \land (z = x \lor z = y)
\]
Synthesis via Counterexample-Guided Instantiation

\[ \neg \exists f. \forall x y. f(x, y) \geq x \land f(x, y) \geq y \land (f(x, y) = x \lor f(x, y) = y) \]

Int \times Int \rightarrow Int

Anti-skolemize

\[ \neg \forall x y. \exists z. z \geq x \land z \geq y \land (z = x \lor z = y) \]

Int

Solve via first-order \( \forall \) techniques

Counterexample Guided QI

[Reynolds et al CAV2015]
Counterexample-Guided $\forall$-Instantiation

Quantifier Elimination Procedures

$\iff (\iff)\ ?$

Instantiation-Based procedures for FO $\exists \forall$ formulas

$\iff$

Synthesis procedures for single-invocation properties
Counterexample-Guided $\forall$-Instantiation: Caveats

1. Specification must be *single invocation*
   - e.g. where functions-to-synthesize are applied to the list of universal variables
     - $\exists f. \forall x y. f(x, y) \geq x \land f(x, y) \geq y$
     - $\exists f g. \forall x. f(x) = g(x)$
     - $\neg \exists f. \forall x y. f(x, y) = f(y, x)$

2. If syntax restrictions are present, CEGQI may violate them
   - Heuristic fitting of solution from CEGQI [Reynolds et al CAV2015]

3. A term selection strategy must be known for the theory
   - Linear arithmetic, small finite domains, BV, ...
   - Strings, non-linear arithmetic, ...
Approach #2:
Smart Enumerative SyGuS
Conjecture

$$\exists f. \forall xy. f(x, y) \geq x \land f(x, y) = f(y, x)$$

Syntactic Restrictions

$$\begin{align*}
\text{fInt} & := x | y | 0 | 1 | +(\text{fInt}, \text{fInt}) | \\
\text{ite} & (\text{fBool}, \text{fInt}, \text{fInt}) \\
\text{fBool} & := \geq (\text{fInt}, \text{fInt}) | = (\text{fInt}, \text{fInt})
\end{align*}$$
Smart Enumerative SyGuS

Conjecture

\[ \exists f. \forall x y. f(x, y) \geq x \land f(x, y) = f(y, x) \]

View syntactic restrictions as an \textit{inductive datatypes}
**Smart Enumerative SyGuS**

**Conjecture**

\[ \exists f. \forall xy. f(x, y) \geq x \land f(x, y) = f(y, x) \]

**Inductive Datatype**

\[
\begin{align*}
\text{fInt} & := x \mid y \mid 0 \mid 1 \mid + \text{(fInt, fInt)} \mid \text{ite} \text{(fBool, fInt, fInt)} \\
\text{fBool} & := \geq \text{(fInt, fInt)} \mid = \text{(fInt, fInt)}
\end{align*}
\]

**Encode using** *deep embedding* **involving** \textbf{fInt}

\[ \exists d. \forall xy. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x) \]

**“Evaluation function”** \( E : \text{fInt} \times \text{Int} \times \text{Int} \rightarrow \text{Int} \)
Smart Enumerative SyGuS

Conjecture

\[ \exists f. \forall xy. f(x, y) \geq x \land f(x, y) = f(y, x) \]

\[ \exists d. \forall xy. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x) \]

Inductive Datatype

\[ f\text{Int} := x | y | 0 | 1 | + (f\text{Int}, f\text{Int}) | \text{ite} (f\text{Bool}, f\text{Int}, f\text{Int}) \]

\[ f\text{Bool} := \geq (f\text{Int}, f\text{Int}) | = (f\text{Int}, f\text{Int}) \]

Solve via datatypes theory solver + CEGIS

Models for \( d \iff \text{candidate solutions} \)
Pruning via Theory Rewriting

Conjecture

\exists d. \forall x y. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x)

Inductive Datatype

\[
\begin{align*}
\text{fInt} := & x \mid y \mid 0 \mid 1 \mid +(\text{fInt}, \text{fInt}) \mid \\
\text{ite} & (\text{fBool}, \text{fInt}, \text{fInt}) \\
\text{fBool} := & \geq(\text{fInt}, \text{fInt}) \mid = (\text{fInt}, \text{fInt})
\end{align*}
\]
Pruning via Theory Rewriting

Conjecture

\( \exists d. \forall x y. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x) \)

Solver generates a stream of candidate models:

- \( d^M = x \)
- \( d^M = y \)
- \( d^M = +(1, y) \)
- \( d^M = +(0, x) \)
- \( d^M = +(y, 1) \)
- ...

Inductive Datatype

\[ f\text{Int} := x \mid y \mid 0 \mid 1 \mid +(f\text{Int}, f\text{Int}) \mid \text{ite}(f\text{Bool}, f\text{Int}, f\text{Int}) \]
\[ f\text{Bool} := \geq(f\text{Int}, f\text{Int}) \mid =(f\text{Int}, f\text{Int}) \]
Pruning via Theory Rewriting

Conjecture

\[ \exists d. \forall x y. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x) \]

Inductive Datatype

\[
\begin{align*}
\text{fInt} & := x \mid y \mid 0 \mid 1 \mid +(\text{fInt}, \text{fInt}) \mid \text{ite}(\text{fBool}, \text{fInt}, \text{fInt}) \\
\text{fBool} & := \geq(\text{fInt}, \text{fInt}) \mid =(\text{fInt}, \text{fInt})
\end{align*}
\]

• Solver generates a stream of candidate models:
  - \( d^M = x \)
  - \( d^M = y \)
  - \( d^M = +(1, y) \)
  - \( d^M = +(0, x) \)
  - \( d^M = +(y, 1) \)

**Optimization:** Only consider terms \( d^M \) whose analog is unique up to theory-specific simplification \( \downarrow \)
Pruning via Theory Rewriting

Conjecture

\[ \exists d. \forall x y. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x) \]

Inductive Datatype

- \( f\text{Int} := x | y | 0 | 1 | +(f\text{Int},f\text{Int}) | \)  
  \( \text{ite}(f\text{Bool}, f\text{Int}, f\text{Int}) \)
- \( f\text{Bool} := \geq(f\text{Int}, f\text{Int}) | = (f\text{Int}, f\text{Int}) \)

- Solver generates a stream of candidate models, normalizes values \( \downarrow \):
  - \( d^M = \ x \) \( \rightarrow \) \( x = \downarrow x \)
  - \( d^M = \ y \) \( \rightarrow \) \( y = \downarrow y \)
  - \( d^M = \ +(1, y) \) \( \rightarrow \) \( 1+y = \downarrow y+1 \)
  - \( d^M = \ +(0, x) \) \( \rightarrow \) \( 0+x = \downarrow x \)
  - \( d^M = \ +(y, 1) \) \( \rightarrow \) \( y+1 = \downarrow y+1 \)
Pruning via Theory Rewriting

Conjecture

\[
\exists d. \forall x, y. E(d, x, y) \geq x \land E(d, x, y) = E(d, y, x)
\]

Inductive Datatype

\[
\begin{align*}
\text{fInt} & := x \mid y \mid 0 \mid 1 \mid +(\text{fInt}, \text{fInt}) \\
\text{fBool} & := \geq(\text{fInt}, \text{fInt}) \mid =\text{(fInt, fInt)}
\end{align*}
\]

• Solver generates a stream of candidate models, normalizes values \(\downarrow\):

  - \(d^M = x\) ... \(x = \downarrow x\)
  - \(d^M = y\) ... \(y = \downarrow y\)
  - \(d^M = +(1, y)\) ... \(1+y = \downarrow y+1\)
  - \(d^M = +(0, x)\) ... \(0+x = \downarrow x\)
  - \(d^M = +(y, 1)\) ... \(y+1 = \downarrow y+1\)

Avoid candidate solutions not unique up to theory normalization
Syntactic Constraints in Smart Enumerative SyGuS

\[-is_+(d) \lor \neg is_x(d.1) \lor \neg is_0(d.2)\]

“Do not consider solutions where \( d \) is \( +(x, 0) \)”

• Encoding uses *shared selectors* of the form \( d.1 \)
  • Agnostic to constructor of \( d \) [Reynolds et al IJCAR 2018]

• Syntactic constraints can be generalized:

\[-is_+(d) \lor \neg is_0(d.2)\]

“Do not consider solutions where \( d \) is of the form \( +(t, 0) \) for any \( t \)”

\( \Rightarrow \) Leads to stronger search space pruning [Reynolds et al CAV 2019]
(Partial) Evaluation Unfolding

\[ \text{is}_{\text{ite}}(d) \Rightarrow E(d, x, y) = \text{ite}(E(d.1, x, y), E(d.2, x, y), E(d.3, x, y)) \]

“When the top symbol of \( d \) is \textit{ite}, its evaluation behaves like if-then-else”

\[ \text{is}_+(d) \land \text{is}_x(d.1) \land \text{is}_1(d.2) \Rightarrow E(d, x, y) = x + 1 \]

“When \( d \) has value \( x + 1 \), its evaluation is equal to \( x + 1 \)”

• Evaluation unfolding lemmas connect evaluation symbols \( E \) to theory
• Implementation combines partial and total unfolding
  • Boolean connectives and ITE use partial unfolding, others use total
Approach #3: Fast Enumerative SyGuS
Fast Enumerative SyGuS

• Directly enumerate terms based on custom iterator data structures
Fast Enumerative SyGuS (vs. Smart)

**PROS:**
- Can use (basic) theory rewriting to prune redundant terms
- Very fast
  - Roughly 100x term throughput w.r.t smart enumeration

**CONS:**
- Cannot easily generalize syntactic constraints
  - Thus, more advanced pruning techniques are not (easily) applicable

**SUMMARY:**
- Fast is usually better, smart is required for harder conjectures [Reynolds et al CAV 2019]
Profiling Fast Enumerative SyGuS

- Evaluating terms on concrete examples is the bottleneck (~70% of runtime)
Summary of Solvers

• If $\Psi$ is single invocation, no grammar restrictions, has theory QI
  (#1) Use *counterexample-guided quantifier instantiation*

• Else:
  • If $\Psi$ has multiple function-to-synthesize or grammar with Bool connectives
    (#2) Use *smart enumerative SyGuS*
  • Else:
    (#3) Use *fast enumerative SyGuS*
Solvers are Supplemented with Additional Techniques

• For CEGQI:
  • Partial quantifier elimination as a preprocessing pass
  • Heuristic solution reconstruction

• For enumerative:
  • Divide-and-conquer [Alur et al 2017]
  • Piecewise-Independent Unification (UNIF+PI) [Barbosa et al FMCAD2019]
  • Theory-specific constant repair [Abate et al 2019]
  • Static grammar minimization and symmetry breaking
  • Variable agnostic enumeration
Ongoing work

• Internal use of SyGuS *for improving the SMT solver*
  • For designing QI algorithms [Niemetz et al CAV 2018, Brain et al CAV2019]
  • Discovering rewrite rules [Noetzli et al SAT 2019]
  • User-guided test case generation
  • Quantifier instantiation via enumerative SyGuS [Niemetz et al TACAS 2021]

• Algorithms that utilize enumerative *SyGuS as a black box*
  • Invariant synthesis [Barbosa et al FMCAD 2019]
  • Abduction [Reynolds et al IJCAR 2020]
  • Interpolation
  • Optimization

• Low-level optimizations
Thanks!

- SyGuS techniques in talk available in SMT solver CVC4(...5)
  - Open-source: https://cvc4.github.io/
  - Includes Python and C++ APIs for SyGuS
  - Java API coming soon

- Questions?