Extending enumerative function synthesis via SMT-driven classification

Haniel Barbosa, Andrew Reynolds, Daniel Larraz, Cesare Tinelli

UFMG

The University of Iowa

Lógicos em Quarentena
2020-04-30, The Internet
Program synthesis: “The Holy Grail of Computer Science”

Specification
High-level description
"What"

Synthesizer

Program
Low-level description
"How"

Extending enumerative function synthesis via SMT-driven classification
Program synthesis: “The Holy Grail of Computer Science”

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

\[ P(x) = \text{ite}(x < 2, x + 1, \text{ite}(x < 3, 2 \times x, 2 \times x + 2)) \]
Program synthesis: “The Holy Grail of Computer Science”

<table>
<thead>
<tr>
<th>Specification</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level description</td>
<td>Low-level description</td>
</tr>
<tr>
<td>&quot;What&quot;</td>
<td>&quot;How&quot;</td>
</tr>
</tbody>
</table>

**Synthesizer**

\[
P(x) = \text{ite}(x < 2, x + 1, \text{ite}(x < 3, 2 \times x, 2 \times x + 2))
\]

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
Program synthesis: “The Holy Grail of Computer Science”

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

\[ P(x) = \text{ite}(x < 2, x + 1, \text{ite}(x < 3, 2 \times x, 2 \times x + 2)) \]

\[ P(x) = 2^x \]
Program synthesis: “The Holy Grail of Computer Science”

### Specification
High-level description
"What"

### Synthesizer

### Program
Low-level description
"How"

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

\[
P(x) = \text{ite}(x < 2, x + 1, \text{ite}(x < 3, 2 \times x, 2 \times x + 2))
\]

\[
P(x) = 2^x
\]

\[
P(x) = \ldots
\]
Program synthesis: “The Holy Grail of Computer Science”

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0, y=1</td>
<td>x=1, y=0</td>
</tr>
<tr>
<td>x=1, y=2</td>
<td>x=2, y=1</td>
</tr>
<tr>
<td>x=2, y=3</td>
<td>x=3, y=2</td>
</tr>
<tr>
<td>x=3, y=4</td>
<td>x=4, y=3</td>
</tr>
</tbody>
</table>
Program synthesis: “The Holy Grail of Computer Science”

```
In  | Out
---|---
x=0, y=1 | x=1, y=0
x=1, y=2 | x=2, y=1
x=2, y=3 | x=3, y=2
x=3, y=4 | x=4, y=3
```

\[ P(x, y) = \{ x \leftarrow x + 1; y \leftarrow y - 1 \} \]
Program synthesis: “The Holy Grail of Computer Science”

- **Specification**
  - High-level description
    - "What"

- **Program**
  - Low-level description
    - "How"

**Synthesizer**

**Main challenges:**
- Exploring search space
- Capturing intention

**Three main characteristics:**
- How to write specification
- How to constrain search space
- How to guide the search

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0, y=1</td>
<td>x=1, y=0</td>
</tr>
<tr>
<td>x=1, y=2</td>
<td>x=2, y=1</td>
</tr>
<tr>
<td>x=2, y=3</td>
<td>x=3, y=2</td>
</tr>
<tr>
<td>x=3, y=4</td>
<td>x=4, y=3</td>
</tr>
</tbody>
</table>

\[
P(x, y) = \{ x \leftarrow x + 1; y \leftarrow y - 1 \}
\]

\[
P(x, y) = \{ z \leftarrow x; x \leftarrow y; y \leftarrow z \}
\]
Program synthesis: “The Holy Grail of Computer Science”

Specification
High-level description
"What"

Synthesizer

Program
Low-level description
"How"

Main challenges:
▶ Exploring search space
▶ Capturing intention

Three main characteristics:
▶ How to write specification
▶ How to constrain search space
▶ How to guide the search

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0,y=1</td>
<td>x=1,y=0</td>
</tr>
<tr>
<td>x=1,y=2</td>
<td>x=2,y=1</td>
</tr>
<tr>
<td>x=2,y=3</td>
<td>x=3,y=2</td>
</tr>
<tr>
<td>x=3,y=4</td>
<td>x=4,y=3</td>
</tr>
</tbody>
</table>

$P(x, y) = \{ x \leftarrow x + 1; y \leftarrow y - 1 \}$

$P(x, y) = \{ z \leftarrow x; x \leftarrow y; y \leftarrow z \}$

$P(x, y) = \ldots$
Program synthesis: “The Holy Grail of Computer Science”

Main challenges:
- Exploring search space
- Capturing intention

Three main characteristics
- How to write specification
- How to constrain search space
- How to guide the search

\[
P(x, y) = \{ x \leftarrow x + 1; y \leftarrow y - 1 \}\]

\[
P(x, y) = \{ z \leftarrow x; x \leftarrow y; y \leftarrow z \}\]

\[
P(x, y) = \ldots\]
Some applications of program synthesis

- Superoptimization
  - [SSA13], [NRB+19], ...

- Program repair
  - [NWK+17], [LCL+17], ...

- Programming by examples
  - [Gul11], [FMG+17] ...

- Circuit synthesis
  - [EWW16], ...

- Loop invariant synthesis
  - [GLM+14], [PSM16], ...

- ...

Extending enumerative function synthesis via SMT-driven classification

2 / 26
Syntax-Guided Synthesis (SyGuS) [ABJ+13]

- Specification
- Syntax restrictions

Synthesizer

Program

- Specification is given by (second-order) $\mathcal{T}$-formula: $\exists f. \forall \bar{x}. \varphi[f, \bar{x}]$

- Syntactic restrictions given by context-free grammar $R$
Enumerative CEGIS

Consider the example:

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]

\[ R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \]

\[ B \rightarrow A \leq A \mid \neg B \]

Counterexamples = 
\{ \}

Solution Enumerator

Solution Verifier

▷ De facto approach to SyGuS solving given its simplicity and efficacy
Enumerative CEGIS

Consider the example:

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]

\[ R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \]

\[ B \rightarrow A \leq A \mid \neg B \]

Counterexamples = 

\{\}  

De facto approach to SyGuS solving given its simplicity and efficacy
Enumerative CEGIS

Consider the example:

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]

\[ R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \]
\[ B \rightarrow A \leq A \mid \neg B \]

Counterexamples =
\{ f(1,1) = 2,
  f(1,2) = 1 \}

\[ \text{Candidate} \quad f(x,y) = x \]

\[ \text{Counterexample} \quad f(x=1,y=0) \]

De facto approach to SyGuS solving given its simplicity and efficacy
Consider the example:

$$\varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x$$

$$R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A)$$

$$B \rightarrow A \leq A \mid \neg B$$

Counterexamples =

\{ f(1,1) = 2,
   f(1,2) = 1 \}  

Examples rule out candidates

0, 1, y, x+y, ...

- De facto approach to SyGuS solving given its simplicity and efficacy
Enumerative CEGIS \[\text{[STB+06; URD+13]}\]

Consider the example:

\[
\varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x
\]

\[
R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A)
\]

\[
B \rightarrow A \leq A \mid \neg B
\]

Counterexamples =

\[
\{ f(1,1) = 2,
     f(1,2) = 1,
     f(0,0) = 1,
     f(0,1) = 0
\}
\]

Candidate

\[
f(x,y)=\text{ite}(y<1, 1+1, 1)
\]

Counterexample

\[
f(x=0,y=0)
\]

De facto approach to SyGuS solving given its simplicity and efficacy
Consider the example:

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]

\[ R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \]
\[ B \rightarrow A \leq A \mid \neg B \]

Counterexamples =
\{ f(1,1) = 2,
  f(1,2) = 1,
  f(0,0) = 1,
  f(0,1) = 0
\}

\[ \triangleright \text{De facto approach to SyGuS solving given its simplicity and efficacy} \]
Scalability issues

Enumerative techniques are effective but limited by the explosion of the enumeration space as term size increases.

For this bit-vector grammar, enumerating:

- Terms of size = 1: .05 seconds
- Terms of size = 2: .6 seconds
- Terms of size = 3: 48 seconds
- Terms of size = 4: 5.8 hours
- Terms of size = 5: ??? (100+ days)
Divide and conquer (D&C) [ARU17; NSM18]

Terms = \{ 0, 1, x, y, x+1 \}

Counterexamples = \{ f(1,1) = 2,
                   f(1,2) = 1,
                   f(0,0) = 1,
                   f(0,1) = 0 \}

Predicate Enumerator

Term Enumerator

Decision Tree Learner

Predicates = \{ y \leq x \}

Solution Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning
Divide and conquer (D&C)

Term Enumerator

Predicate Enumerator

Terms = { 0, 1, x, y, x+1 }

Predicates = { y ≤ x }

Counterexamples = { f(1,1) = 2,
                    f(1,2) = 1,
                    f(0,0) = 1,
                    f(0,1) = 0 }

Decision Tree Learner

Solution Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

$\top$ $y \leq x$ $\bot$

$\{ f(1, 1) = 2, f(0, 0) = 1 \}$

$\{ f(1, 2) = 1, f(0, 1) = 0 \}$
Divide and conquer (D&C)

Terms = 
{ 0, 1, x, y, x+1 }

Counterexamples = 
{ f(1,1) = 2, 
f(1,2) = 1, 
f(0,0) = 1, 
f(0,1) = 0 }

Predicate Enumerator

Term Enumerator

Predicates = 
{ y ≤ x }

Decision Tree Learner

Solution Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

\[
\begin{align*}
\top & \quad y \leq x \\
\{ f(1, 1) = 2, f(0, 0) = 1 \} & \quad \{ f(1, 2) = 1, f(0, 1) = 0 \} \\
\end{align*}
\]
Divide and conquer (D&C) [ARU17; NSM18]

Terms = 
{ 0, 1, x, y, x+1 }

Predicates = 
{ y ≤ x }

Counterexamples = 
{ f(1,1) = 2, 
f(1,2) = 1, 
f(0,0) = 1, 
f(0,1) = 0 }

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

⊥

y ≤ x

{ f(1, 1) = 2, f(0, 0) = 1 }  
{ f(1, 2) = 1, f(0, 1) = 0 }

1 ⊥
Divide and conquer (D&C) [ARU17; NSM18]

Terms =
\{ 0, 1, x, y, x+1 \}

Counterexamples =
\{ f(1,1) = 2, 
f(1,2) = 1, 
f(0,0) = 1, 
f(0,1) = 0 \}

Predicate
Enumerator
Predicates =
\{ y \leq x \}

Decision Tree Learner

Solution Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

\( y \leq x \)

\{ f(1,1) = 2, f(0,0) = 1 \}
\{ f(1,2) = 1, f(0,1) = 0 \}

\( x \neq \)
Divide and conquer (D&C) [ARU17; NSM18]

Terms =
{ 0, 1, x, y, x+1 }

Counterexamples =
{ f(1,1) = 2,
  f(1,2) = 1,
  f(0,0) = 1,
  f(0,1) = 0 }

Predicates =
{ y ≤ x }

Solution
Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

\[ y \leq x \]

\{ f(1, 1) = 2, f(0, 0) = 1 \}
\{ f(1, 2) = 1, f(0, 1) = 0 \}
Divide and conquer (D&C)

Terms =
{ 0, 1, x, y, x+1 }

Counterexamples =
{ f(1,1) = 2,
  f(1,2) = 1,
  f(0,0) = 1,
  f(0,1) = 0 }

Predicate Enumerator

Term Enumerator

Predicates =
{ y \leq x }

Decision Tree Learner

Solution Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

\{ f(1,1) = 2, f(0,0) = 1 \}
\{ f(1,2) = 1, f(0,1) = 0 \}

\( x + 1 \)

\( y \leq x \)
Divide and conquer (D&C) [ARU17; NSM18]

Terms =
{ 0, 1, x, y, x+1 }

Predicate Enumerator

Predicates =
{ y ≤ x }

Counterexamples =
{ f(1,1) = 2,
  f(1,2) = 1,
  f(0,0) = 1,
  f(0,1) = 0 }

Decision Tree Learner

Solution Verifier

▷ Generate partial solutions correct on subset of input
▷ Unify partial solutions via decision tree learning

\[
x + 1 \checkmark
\]

\[
\{ f(1, 1) = 2, f(0, 0) = 1 \}
\]

\[
\{ f(1, 2) = 1, f(0, 1) = 0 \}
\]
Divide and conquer (D&C) [ARU17; NSM18]

Terms = \{ 0, 1, x, y, x+1 \}

Counterexamples = \{ f(1,1) = 2,
                     f(1,2) = 1,
                     f(0,0) = 1,
                     f(0,1) = 0 \}

Predicates = \{ y \leq x \}

Candidate
f(x,y) = \text{ite}(y \leq x, x+1, x)

Decision Tree Learner

Solution Verifier

\begin{itemize}
  \item Generate partial solutions correct on subset of input
  \item Unify partial solutions via decision tree learning
\end{itemize}

\[
\begin{array}{c}
\top & y \leq x & \bot \\
{x + 1} & \checkmark & \{ f(1, 2) = 1, f(0, 1) = 0 \} & x & \checkmark
\end{array}
\]

D&C provides much better scalability
However...

- D&C can only be applied to point-wise specifications
  - Each input valuation is specified independently

Consider augmenting the previous example:

\[ \phi = f(x, x) \simeq x + 1 \land f((x, x + 1)) \simeq x \lor f(x, y) \simeq x + 1 \Rightarrow f(x + 2, y) \simeq x \]

Counterexample \{x \mapsto 1, y \mapsto 0\} yields the constraints:

\[ f(1, 1) \simeq 2 \land f(1, 2) \simeq 1 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \]

\[ \nabla \nabla \] A solution for \( f(1, 0) \) restricts the solution for \( f(3, 0) \)

\[ \nabla \nabla \] Breaks assumption that partial solutions can be found independently

Extending enumerative function synthesis via SMT-driven classification
D&C can only be applied to point-wise specifications
- Each input valuation is specified independently

Consider augmenting the previous example:

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]
\[ \land f(x, y) \simeq x + 1 \Rightarrow f(x + 2, y) \simeq x \]

Counterexample \( \{x \mapsto 1, y \mapsto 0\} \) yields the constraints:

\[ f(1, 1) \simeq 2 \land f(1, 2) \simeq 1 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \]

A solution for \( f(1, 0) \) restricts the solution for \( f(3, 0) \)

Breaks assumption that partial solutions can be found \textit{independently}
Challenges

- This limitation excludes interesting classes of synthesis problems
  - Invariants: $I(x) \land T(x, x') \Rightarrow I(x')$
  - Ranking functions: $\text{rank}(x') < \text{rank}(x)$
  - Modular arithmetic functions: $f(x) \equiv f(x + n)$
  - ...

- Extending D&C to arbitrary (non-point-wise) specifications:
  - Find a term assignment consistent with point dependencies
  - Correctly classify points according to term assignment
This limitation excludes interesting classes of synthesis problems:

- Invariants: \( I(x) \land T(x, x') \Rightarrow I(x') \)
- Ranking functions: \( \text{rank}(x') < \text{rank}(x) \)
- Modular arithmetic functions: \( f(x) \approx f(x + n) \)
- ...

Extending D&C to arbitrary (non-point-wise) specifications:

- Find a term assignment consistent with point dependencies

**SMT solving**

- Correctly classify points according to term assignment
Challenges

- This limitation excludes interesting classes of synthesis problems
  - Invariants: \( I(x) \land T(x, x') \implies I(x') \)
  - Ranking functions: \( \text{rank}(x') < \text{rank}(x) \)
  - Modular arithmetic functions: \( f(x) \simeq f(x + n) \)
  - ...

- Extending D&C to arbitrary (non-point-wise) specifications:
  - Find a term assignment consistent with point dependencies

---

SMT solving

- Correctly classify points according to term assignment

---

Decision tree learning

- SMT-based solution-complete strategy
- Heuristic strategy
SMT solving for SyGuS
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x \mid f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( \mathcal{T}_1, \ldots, \mathcal{T}_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models \mathcal{T}_1, \ldots, \mathcal{T}_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ x_1 \equiv 0 \models \text{LIA} \]

\[ f(x_1) \equiv f(0) \models \text{LIA} \]

\[ x_1 \equiv 0 \models \text{EUF} \]

\[ x_3 + x_1 \not> x_3 + 1 \models \text{EUF} \cup \text{LIA} \]

Therefore \( \models \text{EUF} \cup \text{LIA} \neg \varphi \)

Extending enumerative function synthesis via SMT-driven classification
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x \mid f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an *interpretation* to all terms and predicates such that \( M \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo *equality* and *arithmetic*?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \neq f(0)) \lor x_3 + x_1 > x_3 + 1) \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x \mid f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = \underbrace{(x_1 \geq 0)}_{\text{LIA}} \land \underbrace{(x_1 < 1)}_{\text{LIA}} \land \underbrace{(f(x_1) \not\equiv f(0))}_{\text{EUF}} \lor \underbrace{(x_3 + x_1 > x_3 + 1)}_{\text{LIA}} \]
Satisifiability Modulo Theories (SMT)

First-order formulas

\[ t ::= x \mid f(t, \ldots, t) \]

in CNF:

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an interpretation to all terms and predicates such that \( M \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\cong f(0)) \lor x_3 + x_1 > x_3 + 1 \]

\[ \varphi \models_{\text{LIA}} x_1 \cong 0 \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:
\[ t ::= x \mid f(t, \ldots, t) \]
\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( \mathcal{T}_1, \ldots, \mathcal{T}_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models \mathcal{T}_1, \ldots, \mathcal{T}_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = \begin{cases} (x_1 \geq 0) \land (x_1 < 1) & \text{LIA} \\ (f(x_1) \neq f(0)) & \text{EUF} \\ x_3 + x_1 > x_3 + 1 & \text{LIA} \end{cases} \]

\[ x_1 \simeq 0 \models_{\text{LIA}} \quad x_1 \simeq 0 \]
\[ x_1 \simeq 0 \models_{\text{EUF}} \quad f(x_1) \simeq f(0) \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x \mid f(t, \ldots, t) \]

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an interpretation to all terms and predicates such that \( M \models_{T_1, \ldots, T_n} \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \neq f(0)) \lor x_3 + x_1 > x_3 + 1 \]

\[ \begin{align*}
\varphi & \models_{\text{LIA}} x_1 \simeq 0 \\
x_1 \simeq 0 & \models_{\text{EUF}} f(x_1) \simeq f(0) \\
x_1 \simeq 0 & \models_{\text{LIA}} x_3 + x_1 \not\simeq x_3 + 1
\end{align*} \]
Satisfiability Modulo Theories (SMT)

First-order formulas
\[ t ::= x | f(t, \ldots, t) \]
in CNF:
\[ \varphi ::= p(t, \ldots, t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an interpretation to all terms and predicates such that \( M \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[
\varphi = (x_1 \geq 0) \land (x_1 < 1) \land \underbrace{(f(x_1) \neq f(0)}}_{	ext{EUF}} \lor \underbrace{x_3 + x_1 > x_3 + 1)}_{	ext{LIA}}
\]

\[
\begin{align*}
\varphi & \models \text{LIA} \quad x_1 \simeq 0 \\
x_1 \simeq 0 & \models \text{EUF} \quad f(x_1) \simeq f(0) \\
x_1 \simeq 0 & \models \text{LIA} \quad x_3 + x_1 \not\simeq x_3 + 1
\end{align*}
\]

Therefore
\[
\models \text{EUF} \cup \text{LIA} \quad \neg \varphi
\]
SMT solving

- Decidability depends on the theories being used.

- Efficient decision procedures
  - Equality and uninterpreted functions (Congruence Closure (CC))
    [NO80], [DST80]
  - Algebraic datatypes (CC + Injectivity, Distinctness, Exhaustiveness, Acyclicity)
    [BST07]
  - Linear arithmetic (Simplex)
    [DM06]
  - Bit-vectors (Bit-blasting)
  - Combination of theories (Nelson-Oppen)
  - ... 

- Boolean search leverages SAT solvers

- Users may define their own theories
  - New operators as uninterpreted functions + Axioms
CDCL(\(T\)) architecture

- **Rewriter** simplifies terms
  - \(x + 0 \rightarrow x\)
  - \(a \neq a \rightarrow \bot\)
  - \((\text{str}\_\text{replace} \ x \ (\text{str}\_\text{++} \ x \ x) \ y) \rightarrow x\)

- **SAT solver** enumerates models for Boolean skeleton of formula

- **Theory solvers** check consistency in the theory

- **Instantiation module** selects relevant instances

---

Extending enumerative function synthesis via SMT-driven classification
Enumerative SyGuS in SMT [RKT+17], [RVB+18]

Encode problem using a deep embedding into datatypes

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]

\[ R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \]

\[ B \rightarrow A \leq A \mid \neg B \]

Becomes

\[ \models \varphi = \text{eval}_a(d, x, x) \simeq x + 1 \land \text{eval}_a(d, x, x + 1) \simeq x \]

\[ \models R = \begin{cases} a = \text{Zero} \mid \text{One} \mid X \mid Y \mid \text{Plus}(a, a) \mid \text{Ite}(b, a, a) \\ b = \text{Leq}(a, a) \mid \text{Neg}(b) \end{cases} \]

eval maps datatype terms to their corresponding theory terms

\[ \text{eval}_a(\text{Plus}(X, X), 2, 3) \text{ is interpreted as } (x + x) \{x \mapsto 2, y \mapsto 3\} = 4 \]
Enumerative SyGuS in SMT [RKT+17], [RVB+18]

- Encode problem using a deep embedding into datatypes

\[ \varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \]

\[ R = A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \]

\[ B \rightarrow A \leq A \mid \neg B \]

Becomes

\[ \llbracket \varphi \rrbracket = \text{eval}_a(d, x, x) \simeq x + 1 \land \text{eval}_a(d, x, x + 1) \simeq x \]

\[ \llbracket R \rrbracket = a = \text{Zero} \mid \text{One} \mid x \mid y \mid \text{Plus}(a, a) \mid \text{Ite}(b, a, a) \]

\[ b = \text{Leq}(a, a) \mid \text{Neg}(b) \]

- eval maps datatype terms to their corresponding theory terms

  - \( \text{eval}_a(\text{Plus}(X, X), 2, 3) \) is interpreted as \( (x + x)\{x \mapsto 2, y \mapsto 3\} = 4 \)

- A solution is a model in which e.g.

  - \( d = \text{Ite}(\text{Leq}(Y, X), \text{Plus}(X, \text{One}), X) \), corresponding to
  
  - \( f = \lambda xy. \text{ite}(y \leq x, x + 1, x) \)
An instantiation module checks candidates against the specification
- Generates lemmas witnessing why a candidate failed

A specialized datatypes solver for SyGuS generates candidate solutions
- Must satisfy all lemmas
- Dedicated pruning
- Parameterizable fairness criteria for enumeration
Unif+PI: a general divide-and-conquer framework for SyGuS solving
Recapping

- D&C can only be applied to point-wise specifications
  - Each input valuation is specified independently

- Extending D&C to arbitrary (non-point-wise) specifications requires:
  - Find a term assignment consistent with point dependencies

---

**SMT solving**

- Correctly classify points according to term assignment

---

**Decision tree learning**

- SMT-based solution-complete strategy
- Heuristic strategy
Unif+PI: Synthesis via Pointwise-Independent unification

▷ **SMT-based classifier**
  - Assigns terms to points so that lemmas hold
    \[
    f(1, 1) \mapsto y + y, \quad \{f(1, 0), f(3, 0), f(1, 2)\} \mapsto x
    \]
  - Generates ordered list of predicates to separate points: \(P_1 \mapsto x \neq y\)

▷ **Classification checker**: whether corresponding decision tree correctly classifies sample
  - Failures are encoded as *separation lemmas*
Unif+PI: Synthesis via Pointwise-Independent unification

- **SMT-based Classifier**
- **Classification checker**
- **Verification oracle**

- **Term assignment**
- **Ordered predicates list**
- **Candidate solution**
- **Separation lemmas**
- **Refinement lemmas**

- Successful candidates that are not verified lead to refinement lemmas and the learning restarts
Unif+PI: Synthesis via Pointwise-Independent unification

- Successful candidates that are not verified lead to refinement lemmas and the learning restarts
- Bounded solution-completeness and minimality results due to exhaustive enumeration of possible classifiers according to
  - size and number of distinct terms to be assigned
  - size and number of distinct predicates

Extending enumerative function synthesis via SMT-driven classification
Successful candidates that are not verified lead to refinement lemmas and the learning restarts.

Bounded solution-completeness and minimality results due to exhaustive enumeration of possible classifiers according to

- size and number of distinct terms to be assigned
- size and number of distinct predicates

Our fairness criteria are size = \(\log_2(\#\text{terms})\), \#\text{pred} = \#\text{terms} - 1
Consider again:

$$\varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x$$
$$\land f(x, y) \simeq x + 1 \Rightarrow f(x + 2, y) \simeq x$$

- Initially a single term of size 0 will be a trivial successful classifier
Consider again:
\[
\varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \\
\land f(x, y) \simeq x + 1 \implies f(x + 2, y) \simeq x
\]

▶ Initially a single term of size 0 will be a trivial successful classifier

▶ Refinement lemma:
\[
f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \implies f(3, 0) \simeq 1 \land f(1, 2) \simeq 1
\]
Consider again:

$$\varphi = f(x, x) \sim x + 1 \land f(x, x + 1) \sim x \land f(x, y) \sim x + 1 \Rightarrow f(x + 2, y) \sim x$$

- Initially a single term of size 0 will be a trivial successful classifier
- Refinement lemma:
  $$f(1, 1) \sim 2 \land f(1, 0) \sim 2 \Rightarrow f(3, 0) \sim 1 \land f(1, 2) \sim 1$$
- Since no assignment with a single term suffices, the threshold is increased to consider two distinct terms
  - Maximum size increases to 1 and up to 1 predicate can be used
Consider again:
\[
\varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \\
\land f(x, y) \simeq x + 1 \Rightarrow f(x + 2, y) \simeq x
\]

- Initially a single term of size 0 will be a trivial successful classifier

- Refinement lemma:
  \[
  f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \land f(1, 2) \simeq 1
  \]

- Since no assignment with a single term suffices, the threshold is increased to consider two distinct terms
  - Maximum size increases to 1 and up to 1 predicate can be used

- A candidate classifier is
  \[
  f(1, 1) \mapsto y + y, \quad \{f(1, 0), f(3, 0), f(1, 2)\} \mapsto x \\
  P_1 \mapsto \top
  \]
Consider again:

\[
\varphi = f(x, x) \simeq x + 1 \land f(x, x + 1) \simeq x \\
\land f(x, y) \simeq x + 1 \Rightarrow f(x + 2, y) \simeq x
\]

- Initially a single term of size 0 will be a trivial successful classifier

- Refinement lemma:

\[
f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \land f(1, 2) \simeq 1
\]

- Since no assignment with a single term suffices, the threshold is increased to consider two distinct terms
  - Maximum size increases to 1 and up to 1 predicate can be used

- A candidate classifier is

\[
f(1, 1) \mapsto y + y, \quad \{f(1, 0), f(3, 0), f(1, 2)\} \mapsto x \\
P_1 \mapsto \top
\]

- This classifier fails on the sample, yielding a separation lemma

\[
P_1 \simeq \top \Rightarrow f(1, 1) \simeq f(1, 0)
\]
\[ \varphi_R = f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \land f(1, 2) \simeq 1 \]

\[ \varphi_S = P_1 \simeq \top \Rightarrow f(1, 1) \simeq f(1, 0) \]

Given this constraints and current threshold the next candidate classifier produced is:

\[ \{ f(1, 1), f(1, 0), f(3, 0) \} \mapsto y + 1, \quad f(1, 2) \mapsto 1 \]

\[ P_1 \mapsto y \leq x \]
\[ \varphi_R = f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \implies f(3, 0) \simeq 1 \land f(1, 2) \simeq 1 \]

\[ \varphi_S = P_1 \simeq \top \implies f(1, 1) \simeq f(1, 0) \]

Given this constraints and current threshold the next candidate classifier produced is:

\[ \{ f(1, 1), f(1, 0), f(3, 0) \} \mapsto y + 1, \quad f(1, 2) \mapsto 1 \]

\[ P_1 \mapsto y \leq x \]

Running the classification checker:

\[ f(1, 1) \]
\[ \varphi_R = f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \land f(1, 2) \simeq 1 \]

\[ \varphi_S = P_1 \simeq \top \Rightarrow f(1, 1) \simeq f(1, 0) \]

- Given this constraints and current threshold the next candidate classifier produced is:

\[
\{ f(1, 1), f(1, 0), f(3, 0) \} \mapsto y + 1, \quad f(1, 2) \mapsto 1
\]

\[
P_1 \mapsto y \leq x
\]

- Running the classification checker:

\[
f(1, 1), f(1, 0)
\]
\[ \varphi_R = f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \land f(1, 2) \simeq 1 \]
\[ \varphi_S = P_1 \simeq \top \Rightarrow f(1, 1) \simeq f(1, 0) \]

- Given this constraints and current threshold the next candidate classifier produced is:
  \[ \{ f(1, 1), f(1, 0), f(3, 0) \} \mapsto y + 1, \quad f(1, 2) \mapsto 1 \]
  \[ P_1 \mapsto y \leq x \]

- Running the classification checker:

  \[ f(1, 1), f(1, 0), f(3, 0) \]
Given this constraints and current threshold the next candidate classifier produced is:

\[
\{ f(1, 1), f(1, 0), f(3, 0) \} \mapsto y + 1, \quad f(1, 2) \mapsto 1
\]

Running the classification checker:

\[
f(1, 1), f(1, 0), f(3, 0) \triangleq f(1, 2)
\]
Given this constraints and current threshold the next candidate classifier produced is:

\[ \{ f(1, 1), f(1, 0), f(3, 0) \} \mapsto y + 1, \quad f(1, 2) \mapsto 1 \]

\[ P_1 \mapsto y \leq x \]

Running the classification checker:

\[ f(1, 1), f(1, 0), f(3, 0) \diamond f(1, 2) \rightarrow \]

\[ \top \]

\[ \bot \]

\[ f(1, 1), f(1, 0), f(3, 0) \quad f(1, 2) \]
\[ \varphi_R = f(1, 1) \simeq 2 \land f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \land f(1, 2) \simeq 1 \]
\[ \varphi_S = P_1 \simeq \top \Rightarrow f(1, 1) \simeq f(1, 0) \]

Given this constraints and current threshold the next candidate classifier produced is:

\[ \{f(1, 1), f(1, 0), f(3, 0)\} \mapsto y + 1, \quad f(1, 2) \mapsto 1 \]
\[ P_1 \mapsto y \leq x \]

Running the classification checker:

\[ f(1, 1), f(1, 0), f(3, 0) \diamond f(1, 2) \mapsto \]
\[ \top \]
\[ \bot \]

As the classification succeeds, a candidate is generated

The candidate fails, so the process restarts with new refinement lemmas

Eventually finds solution \( f = \lambda x y. \text{ite}(x \leq y, \text{ite}(y \leq x, x + 1, x), y) \)
Unif+PI with unconstrained predicate enumeration

- **Unif+PI+E** uses SMT solver only to produce term assignments
  - Relies on standard decision tree learning to classify a labeled sample
  - Predicates chosen from enumerated pool with information-gain heuristic
  - Separation conflicts solved when new predicates are enumerated

- Often sacrificing completeness and minimality allows problems to be solved more efficiently
Experimental results
Setup

- **Benchmarks (all over LIA)**
  - 127 invariant synthesis benchmarks from SyGuS-COMP’18
  - 440 invariant synthesis benchmarks from test suite of Kind 2

- **Three configurations of CVC4SY**
  - CVC+C
  - CVC+UPI
  - CVC+UPI+E
  - enumerative CEGIS [RBN+19]
  - Unif+PI
  - Unif+PI+E

- LOOPINVGEN [PM17] and CVC+C as baselines

- 1800s timeout, 8gb RAM

## Summary

<table>
<thead>
<tr>
<th></th>
<th>Solved</th>
<th>Unique</th>
<th>Total time</th>
<th>Fastest</th>
<th>Shortest</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CVC+C</strong></td>
<td>341</td>
<td>30</td>
<td>436251s</td>
<td>245</td>
<td>259</td>
</tr>
<tr>
<td><strong>LOOPINVGEN</strong></td>
<td>298</td>
<td>7</td>
<td>433273s</td>
<td>261</td>
<td>289</td>
</tr>
</tbody>
</table>

![Graph showing the comparison between cvc+c and loopinvgen](chart.png)
<table>
<thead>
<tr>
<th>Method</th>
<th>Solved</th>
<th>Unique</th>
<th>Total time</th>
<th>Fastest</th>
<th>Shortest</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC+C</td>
<td>341</td>
<td>30</td>
<td>436251s</td>
<td>245</td>
<td>259</td>
</tr>
<tr>
<td>CVC+UPI</td>
<td>291</td>
<td>3</td>
<td>494534s</td>
<td>236</td>
<td>231</td>
</tr>
<tr>
<td>LOOPINVGEN</td>
<td>298</td>
<td>7</td>
<td>433273s</td>
<td>261</td>
<td>289</td>
</tr>
</tbody>
</table>

Extending enumerative function synthesis via SMT-driven classification
## Summary

<table>
<thead>
<tr>
<th></th>
<th>Solved</th>
<th>Unique</th>
<th>Total time</th>
<th>Fastest</th>
<th>Shortest</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC+C</td>
<td>341</td>
<td>30</td>
<td>436251s</td>
<td>245</td>
<td>259</td>
</tr>
<tr>
<td>CVC+UPI+E</td>
<td>332</td>
<td>47</td>
<td>414356s</td>
<td>306</td>
<td>222</td>
</tr>
<tr>
<td>CVC+UPI</td>
<td>291</td>
<td>3</td>
<td>494534s</td>
<td>236</td>
<td>231</td>
</tr>
<tr>
<td>LOOPINVGEN</td>
<td>298</td>
<td>7</td>
<td>433273s</td>
<td>261</td>
<td>289</td>
</tr>
<tr>
<td>CVC-PORT</td>
<td>400</td>
<td>-</td>
<td>31476s</td>
<td>379</td>
<td>306</td>
</tr>
</tbody>
</table>

Extending enumerative function synthesis via SMT-driven classification
Advantages and disadvantages of Unif+PI

▷ CVC+UPI and CVC+UPI+E thrive when invariants can be built from combination of small literals

▷ CVC+C is superior when invariant is a single complex literal
  ▶ 29 of its 30 unique solves are such cases

▷ CVC+UPI and CVC+UPI+E also suffer from dependence on samples
  ▶ Sometimes search is biased towards simple classifiers when only a more complex one would suffice
### Inv Track (829)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>Fastest</th>
<th>Smallest</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC4-su</td>
<td>592</td>
<td>423</td>
<td>264</td>
<td>4493</td>
</tr>
<tr>
<td>LoopInvGen</td>
<td>512</td>
<td>442</td>
<td>364</td>
<td>4250</td>
</tr>
<tr>
<td>LoopInvGen-gplearn</td>
<td>511</td>
<td>411</td>
<td>349</td>
<td>4137</td>
</tr>
<tr>
<td>CVC4-Fast</td>
<td>522</td>
<td>319</td>
<td>243</td>
<td>3810</td>
</tr>
<tr>
<td>CVC4-Smart</td>
<td>539</td>
<td>283</td>
<td>260</td>
<td>3804</td>
</tr>
<tr>
<td>OASIS</td>
<td>538</td>
<td>20</td>
<td>317</td>
<td>3067</td>
</tr>
<tr>
<td>DryadSynth</td>
<td>277</td>
<td>161</td>
<td>39</td>
<td>1907</td>
</tr>
</tbody>
</table>

- 829 benchmarks from the literature in loop invariant synthesis
- 3600s timeout
Injecting some welcome realism

- Kind 2 employs in cooperation:
  - IC3 [Bra11]
  - $k$-induction [SSS00]
  - Generation of auxiliary invariants [KGT11]

- Kind 2 solves all the 480 benchmarks it its test suite in less than 120s

- Considering $k$-induction in isolation, CVC-PORT is competitive

<table>
<thead>
<tr>
<th></th>
<th>Solved</th>
<th>Unique</th>
<th>Time (commonly solved)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC-PORT</td>
<td>323</td>
<td>82</td>
<td>109.6</td>
</tr>
<tr>
<td>Kind 2 ($k$-induction)</td>
<td>313</td>
<td>72</td>
<td>9.6</td>
</tr>
</tbody>
</table>
Injecting some welcome realism

- Kind 2 employs in cooperation:
  - IC3
  - $k$-induction
  - Generation of auxiliary invariants

- Kind 2 solves all the 480 benchmarks it its test suite in less than 120s

- Considering $k$-induction in isolation, CVC-PORT is competitive

<table>
<thead>
<tr>
<th></th>
<th>Solved</th>
<th>Unique</th>
<th>Time (commonly solved)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC-PORT</td>
<td>323</td>
<td>82</td>
<td>109.6</td>
</tr>
<tr>
<td>Kind 2 ($k$-induction)</td>
<td>313</td>
<td>72</td>
<td>9.6</td>
</tr>
</tbody>
</table>

- We consider this encouraging given our framework is
  - not theory-specific
  - single-threaded
  - not optimized for reachability
Conclusions
Conclusions

- New enumerative function synthesis framework via divide and conquer
  - No dependence on point-wise specifications
  - Powered by SMT-driven classification algorithms
  - Implemented in CVC4SY

- Experimental evaluation shows significant gains w.r.t. previous SyGuS techniques for invariant synthesis
Future work

- Improving classification
  - Using constraint solving for synthesizing term assignments
  - Only considering relevant arguments when synthesizing predicates
    \[ f(0, 0, 0, 1, 2, 1, 0) \land f(1, 0, 0, 5, 2, 1, 3) \]
    - Can drastically reduce search space

- Improving sample
  - Reducing noise: make points as similar as possible
    \[ f(1, 0, 0, 1, 2, 1, 0) \land f(1, 0, 0, 5, 2, 1, 0) \]
  - Improve diversity via clustering analysis: only add new points to sample that are sufficiently different
Extending enumerative function synthesis via SMT-driven classification

Haniel Barbosa, Andrew Reynolds, Daniel Larraz, Cesare Tinelli

UFMG, The University of Iowa

Lógicos em Quarentena
2020-04-30, The Internet
Extra slides
Invariant Synthesis

Add(Int x, y) {
  z := x; i := 0;
  assume(y > 0);
  while (i < y) {
    z := z + 1;
    i := i + 1;
  }
  return z;
}

Post-condition: Result is the sum of the inputs
\( \forall x, y : z = x + y \)
Invariant Synthesis

Add(Int x, y) {
    z := x; i := 0;
    assume(y > 0);
    while (i < y) {
        z := z + 1;
        i := i + 1;
    }
    return z;
}

Post-condition:
\( \forall x, y : z = x + y \)

Verification:

\[
\begin{align*}
    z &= x \land i = 0 \land y > 0 \\
    Inv(x, y, z, i) \land i < y \land z' &= z + 1 \land i' = i + 1 \\
    Inv(x, y, z, i) \land i \geq y \\
    \rightarrow & \quad Inv(x, y, z, i) \\
    \rightarrow & \quad Inv(x, y, z', i') \\
    \rightarrow & \quad z = x + y
\end{align*}
\]
Invariant Synthesis

Add(Int x, y) {
  z := x; i := 0;
  assume(y > 0);
  while (i < y) {
    z := z + 1;
    i := i + 1;
  }
  return z;
}

Post-condition: 
\[ \forall x, y : z = x + y \]

Verification:

\[
\begin{align*}
  z & = x \land i = 0 \land y > 0 \\
  Inv(x, y, z, i) \land i < y \land z' = z + 1 \land i' = i + 1 & \Rightarrow Inv(x, y, z', i') \\
  Inv(x, y, z, i) \land i \geq y & \Rightarrow z = x + y
\end{align*}
\]
Invariant Synthesis in SyGuS

- State-of-the-art: LoopInvGen [PM17]: *data-driven* loop invariant inference with automatic feature synthesis
  - Precondition inference from sets of “good” and “bad” states
    - Feature synthesis for solving conflicts
  - PAC (*probably approximately correct*) algorithm for building candidate invariants

- “Bad” states are dependent on model of initial condition (no guaranteed convergence)

- No support for implication counterexamples
Invariant Synthesis with Unif+PI

- Refinement lemmas allows derivation of three kinds on data points:
  - “good points” (invariant must always hold)
  - “bad points” (invariant can never hold)
  - “implication points” (if invariant holds in first point it must hold in second)

- Native support for implication counterexamples

- Straightforward usage of classic information gain heuristic to build candidate solutions with decision tree learning
  - SMT solver “resolves” implication counterexample points as “good” and “bad”
  - Out-of-the-box Shannon entropy
References


References


