Extending enumerative function synthesis via SMT-driven classification

Haniel Barbosa, Andrew Reynolds, Daniel Larraz, Cesare Tinelli

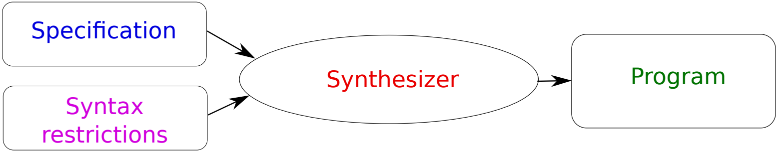
UFMG

The University of Iowa

FMCAD 2019

2019-10-25, San Jose, CA, USA

Syntax-Guided Synthesis (SyGuS) [Alur et al. 2013]



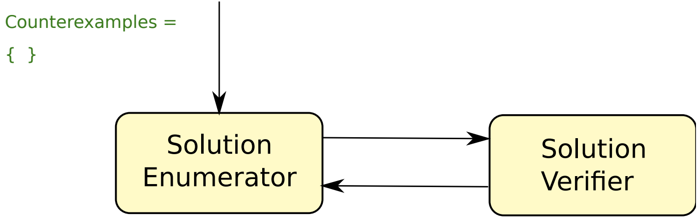
- ▷ Specification is given by T-formula: $\exists f.\forall \bar{x}.\ \varphi[f,\bar{x}]$
- ▷ Syntactic restrictions given by context-free grammar R

Enumerative CEGIS [Solar-Lezama et al. 2006; Udupa et al. 2013]

Consider the example:

$$\varphi \quad = \quad f(x, x) \simeq x + 1 \;\wedge\; f(x, x+1) \simeq x$$

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \\ B \to A \leq A \mid \neg B \end{array}$$



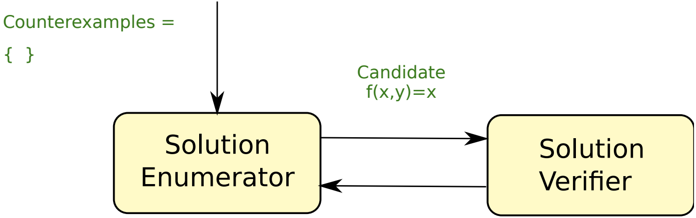
- ▷ De facto approach to SyGuS solving given its simplicity and efficacy

Enumerative CEGIS [Solar-Lezama et al. 2006; Udupa et al. 2013]

Consider the example:

$$\varphi \quad = \quad f(x, x) \simeq x + 1 \;\wedge\; f(x, x+1) \simeq x$$

$$R \quad = \quad \begin{array}{l} A \rightarrow 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B, A, A) \\ B \rightarrow A \leq A \mid \neg B \end{array}$$

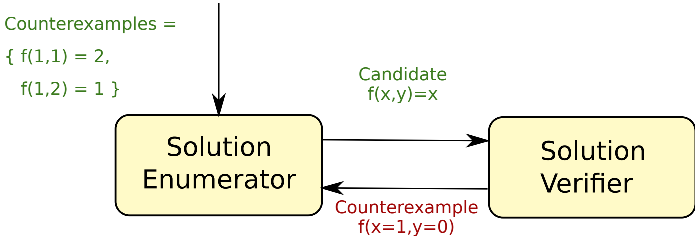


▷ De facto approach to SyGuS solving given its simplicity and efficacy

Consider the example:

$$\varphi \quad = \quad f(x,\, x) \simeq x + 1 \;\wedge\; f(x,\, x+1) \simeq x$$

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B,\, A,\, A) \\ B \to A \leq A \mid \neg B \end{array}$$



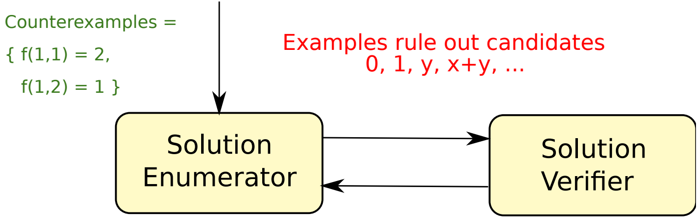
▷ De facto approach to SyGuS solving given its simplicity and efficacy

Enumerative CEGIS [Solar-Lezama et al. 2006; Udupa et al. 2013]

Consider the example:

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

$$R = \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \\ B \to A \leq A \mid \neg B \end{array}$$



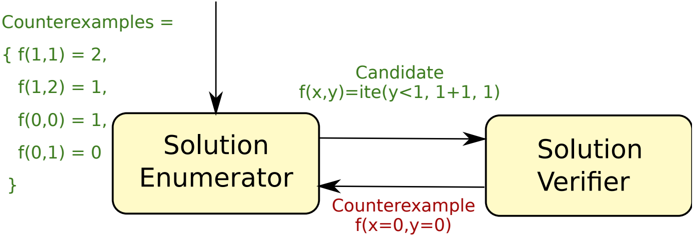
▷ De facto approach to SyGuS solving given its simplicity and efficacy

Enumerative CEGIS [Solar-Lezama et al. 2006; Udupa et al. 2013]

Consider the example:

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

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B, A, A) \\ B \to A \leq A \mid \neg B \end{array}$$



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

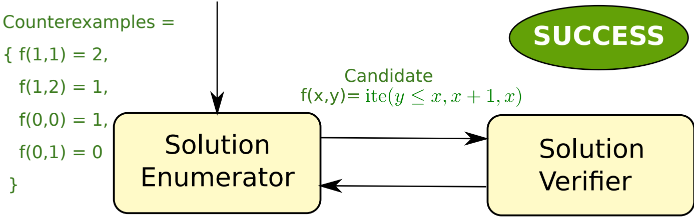
Enumerative CEGIS

[Solar-Lezama et al. 2006; Udupa et al. 2013]

Consider the example:

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

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B,\, A,\, A) \\ B \to A \leq A \mid \neg B \end{array}$$



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

Scalability issues

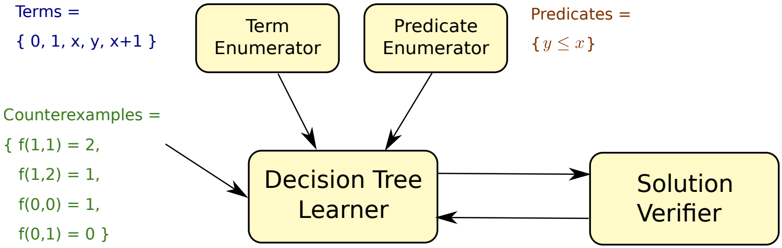
Enumerative techniques are effective but limited to the generation of small terms due to the explosion of the space of terms as size increases

For this bit-vector grammar, enumerating

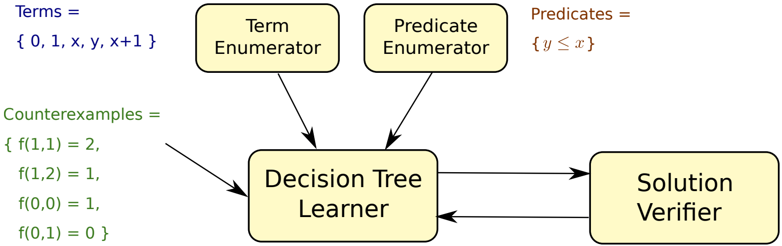
- Terms of size = 1 : .05 seconds
- Terms of size = 2 : .6 seconds
- Terms of size = 3 : 48 seconds
- Terms of size = 4 : 5.8 hours
- Terms of size = 5 : ??? (100+ days)

```
(synth-fun f ((s (BitVec 4))
              (t (BitVec 4)))
 (BitVec 4) (
 (Start (BitVec 4) (
  s t #x0
  (bvneg  Start)
  (bvnot  Start)
  (bvadd  Start Start)
  (bvmul  Start Start)
  (bvand  Start Start)
  (bvlshr Start Start)
  (bvor   Start Start)
  (bvshl  Start Start)))))
```

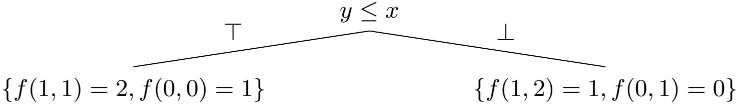
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



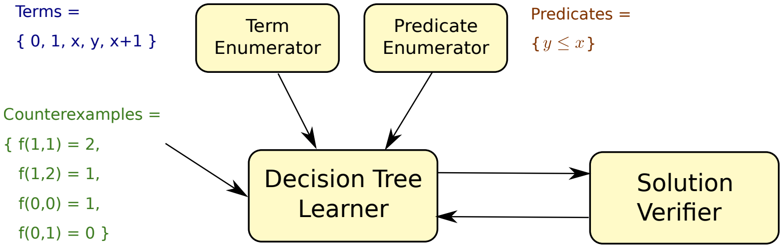
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



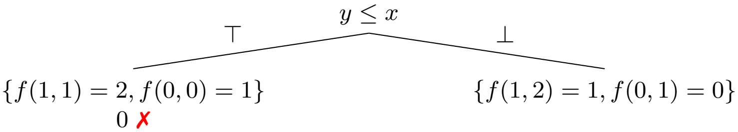
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



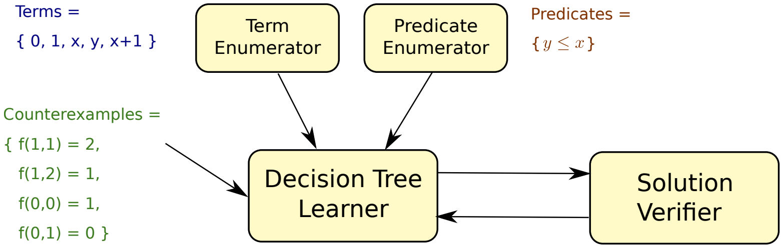
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



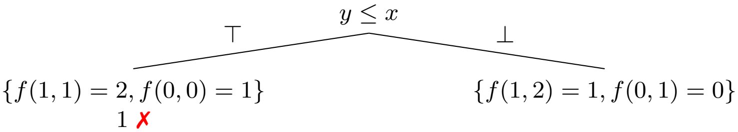
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



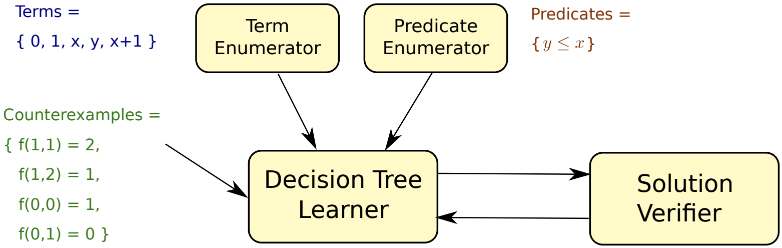
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



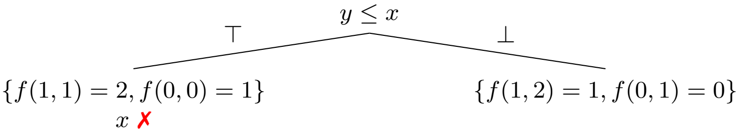
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



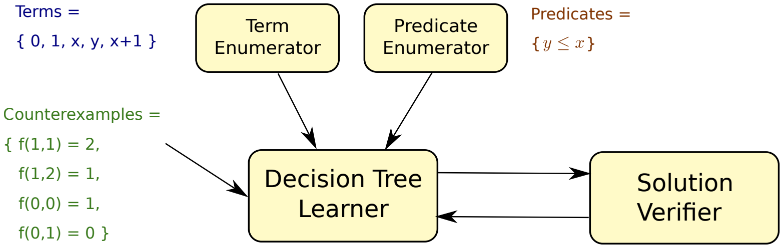
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



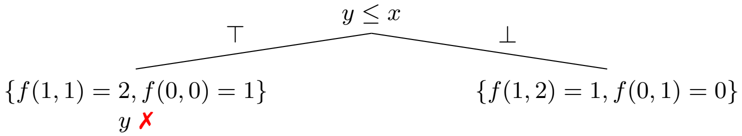
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



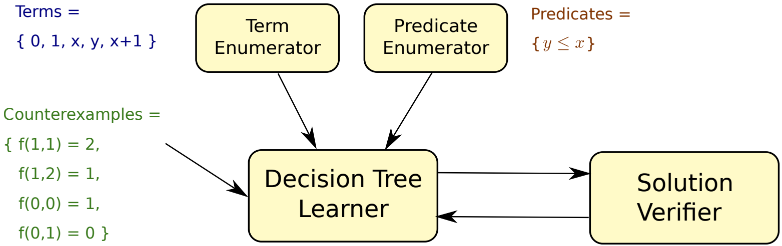
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



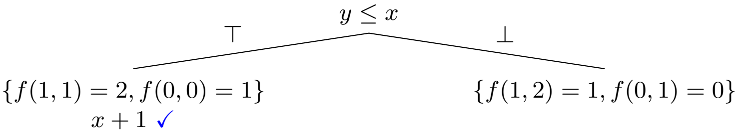
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



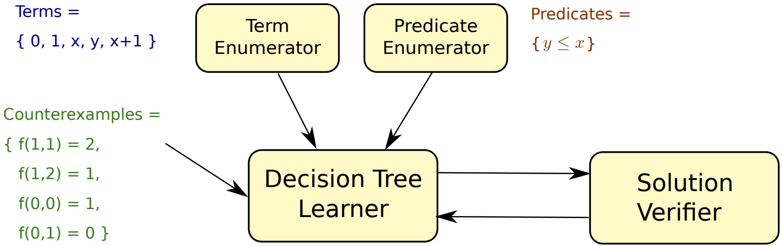
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



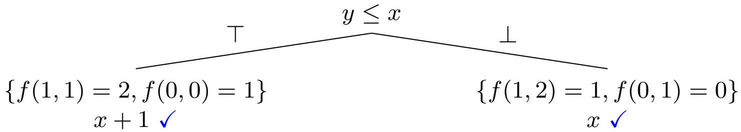
- ▷ Generate partial solutions correct on subset of input
- ▷ Unify partial solutions via decision tree learning



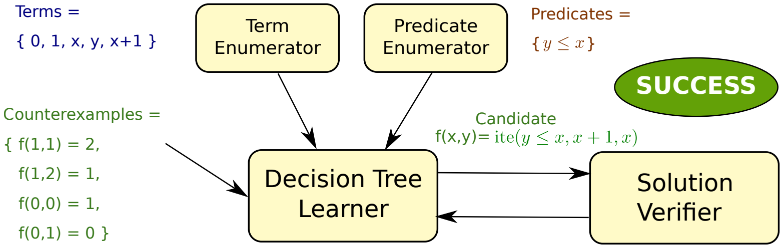
Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



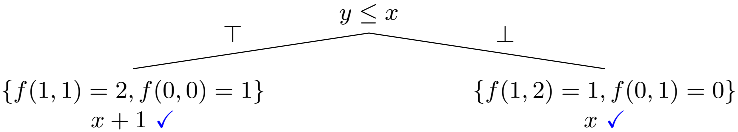
- Generate partial solutions correct on subset of input
- Unify partial solutions via decision tree learning



Divide and conquer (D&C) [Alur et al. 2017; Neider et al. 2018]



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



- ▷ D&C provides much better scalability

However...

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

However...

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

Consider augmenting the previous example:

$$\begin{aligned}\varphi = \quad & f(x, x) \simeq x + 1 \wedge f(x, x+1) \simeq x \\ \wedge \quad & f(x, y) \simeq x + 1 \Rightarrow f(x+2, y) \simeq x\end{aligned}$$

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

$$f(1, 1) \simeq 2 \quad \wedge \quad f(1, 2) \simeq 1 \quad \wedge \quad f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1$$

- A solution for $f(1, 0)$ restricts the solution for $f(3, 0)$
- Breaks assumption that partial solutions can be found *indepedently*

Challenges

- This limitation excludes interesting classes of synthesis problems
  - Invariants: $I(x) \wedge T(x, x') \Rightarrow I(x')$
  - Ranking functions: $rank(x') < rank(x)$
  - 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
  - Correctly classify points according to term assignment

Challenges

- This limitation excludes interesting classes of synthesis problems
  - Invariants: $I(x) \wedge T(x, x') \Rightarrow I(x')$
  - Ranking functions: $rank(x') < rank(x)$
  - 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

Challenges

- This limitation excludes interesting classes of synthesis problems
  - Invariants: $I(x) \wedge T(x, x') \Rightarrow I(x')$
  - Ranking functions: $rank(x') < rank(x)$
  - 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

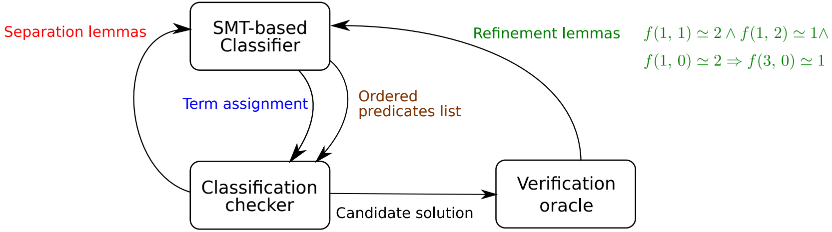
  - Correctly classify points according to term assignment

Decision tree learning

- SMT-based solution-complete strategy
- Heuristic strategy

Unif+PI: a general divide-and-conquer framework for SyGuS solving

Unif+PI: Synthesis via Pointwise-Indepentent unification

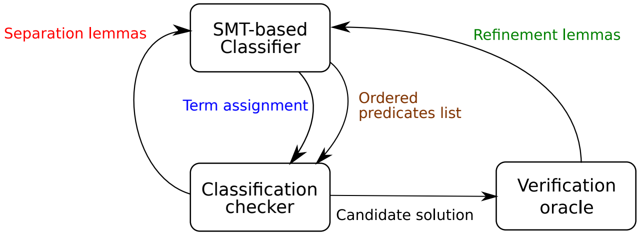


- *SMT-based classifier*
  - Assigns terms to points so that lemmas hold

$$f(1, 1) \mapsto y + y, \quad \{f(1, 0), f(3, 0), f(1, 2)\} \mapsto x$$

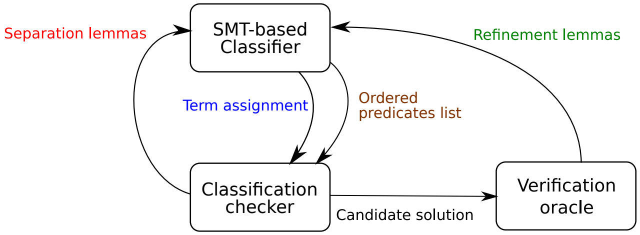
  - Generates ordered list of predicates to *separate* points: $P_1 \mapsto x \neq y$
- *Classification checker*: whether corresponding decision tree correctly classifies sample
  - Failures are encoded as *separation lemmas*

Unif+PI: Synthesis via Pointwise-Indepentent unification



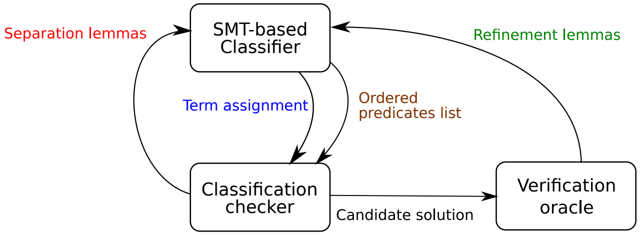
- ▷ Successful candidates that are not verified lead to refinement lemmas and the learning restarts

Unif+PI: Synthesis via Pointwise-Indepentent 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

Unif+PI: Synthesis via Pointwise-Indepentent 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
- ▷ Our fairness criteria are $size = log_2(\#terms)$, $\#pred = \#terms - 1$

Consider again:

$$\begin{aligned}\varphi = \quad & f(x,\, x) \simeq x + 1 \,\wedge\, f(x,\, x+1) \simeq x \\ \wedge \quad & f(x,\, y) \simeq x + 1 \Rightarrow f(x+2,\, y) \simeq x\end{aligned}$$

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

Consider again:

$$\begin{aligned} \varphi = \quad & f(x, x) \simeq x + 1 \wedge f(x, x+1) \simeq x \\ \wedge \quad & f(x, y) \simeq x + 1 \Rightarrow f(x+2, y) \simeq x \end{aligned}$$

- ▷ Initially a single term of size 0 will be a trivial successful classifier
- ▷ Refinement lemma:

$$f(1, 1) \simeq 2 \quad \wedge \quad f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \quad \wedge \quad f(1, 2) \simeq 1$$

Consider again:

$$\begin{aligned}\varphi = \quad & f(x, x) \simeq x + 1 \wedge f(x, x+1) \simeq x \\ \wedge \quad & f(x, y) \simeq x + 1 \Rightarrow f(x+2, y) \simeq x\end{aligned}$$

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

$$f(1, 1) \simeq 2 \quad \wedge \quad f(1, 0) \simeq 2 \Rightarrow f(3, 0) \simeq 1 \quad \wedge \quad f(1, 2) \simeq 1$$

- 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

Consider again:

$$\begin{aligned} \varphi = \quad & f(x,\, x) \simeq x+1 \wedge f(x,\, x+1) \simeq x \\ \wedge \quad & f(x,\, y) \simeq x+1 \Rightarrow f(x+2,\, y) \simeq x \end{aligned}$$

- ▷ Initially a single term of size 0 will be a trivial successful classifier
- ▷ Refinement lemma:

$$f(1,\, 1) \simeq 2 \quad \wedge \quad f(1,\, 0) \simeq 2 \Rightarrow f(3,\, 0) \simeq 1 \quad \wedge \quad f(1,\, 2) \simeq 1$$

- ▷ 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:

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

- ▷ Initially a single term of size 0 will be a trivial successful classifier
- ▷ Refinement lemma:

$$f(1,\,1) \simeq 2 \quad \wedge \quad f(1,\,0) \simeq 2 \Rightarrow f(3,\,0) \simeq 1 \quad \wedge \quad f(1,\,2) \simeq 1$$

- ▷ 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

$$P_1 \simeq \top \Rightarrow f(1,\,1) \simeq f(1,\,0)$$

$$\begin{aligned}\varphi_R &= f(1,1) \simeq 2 \quad \wedge \quad f(1,0) \simeq 2 \Rightarrow f(3,0) \simeq 1 \quad \wedge \quad f(1,2) \simeq 1 \\ \varphi_S &= P_1 \simeq \top \Rightarrow f(1,1) \simeq f(1,0)\end{aligned}$$

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

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

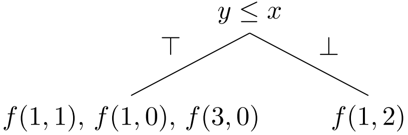
$$
\begin{aligned}
\varphi_R &= f(1,\,1) \simeq 2 \quad \wedge \quad f(1,\,0) \simeq 2 \Rightarrow f(3,\,0) \simeq 1 \quad \wedge \quad f(1,\,2) \simeq 1 \\
\varphi_S &= P_1 \simeq \top \Rightarrow f(1,\,1) \simeq f(1,\,0)
\end{aligned}
$$

- ▷ 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 ClassChecker:

$f(1,1),\, f(1,0),\, f(3,0) \diamond f(1,2) \;\rightarrow$



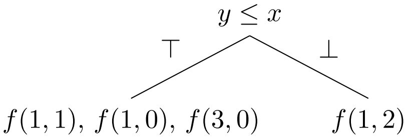
$$\varphi_R = f(1,1) \simeq 2 \quad \wedge \quad f(1,0) \simeq 2 \Rightarrow f(3,0) \simeq 1 \quad \wedge \quad f(1,2) \simeq 1$$
$$\varphi_S = P_1 \simeq \top \Rightarrow f(1,1) \simeq f(1,0)$$

- 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 ClassChecker:

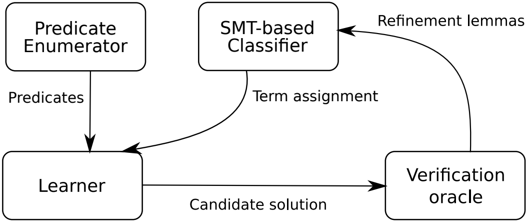
$f(1,1),\ f(1,0),\ f(3,0) \diamond f(1,2) \ \rightarrow$



- As the classification succeeds, a candidate is generated

- But this candidate fails, so the process restarts with new refinement lemmas

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

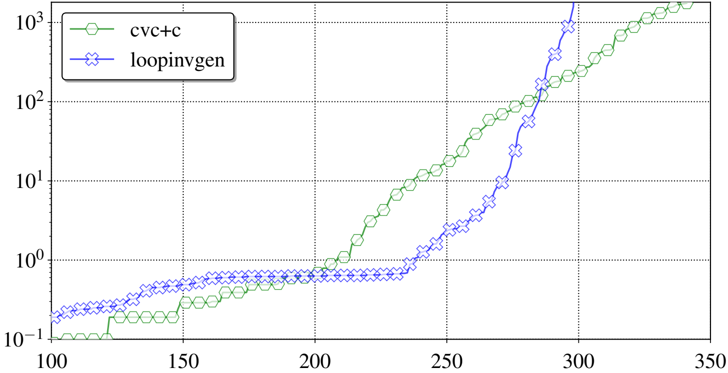
CVC+C	enumerative CEGIS [Reynolds et al. 2019]
CVC+UPI	Unif+PI
CVC+UPI+E	Unif+PI+E

- LOOPINVGEN [Padhi and Millstein 2017] and CVC+C as baselines
- 1800s timeout, 8gb RAM

Full data at `http://cvc4.cs.stanford.edu/papers/FMCAD2019-UnifPI/`

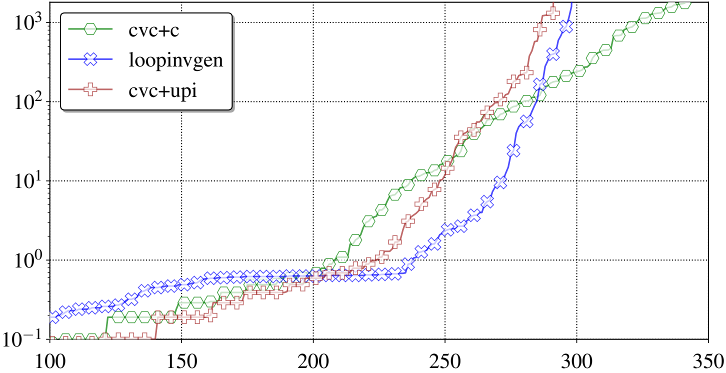
Summary

	Solved	Unique	Total time	Fastest	Shortest
CVC+C	**341**	30	436251s	245	259
LOOPINVGEN	298	7	433273s	261	**289**



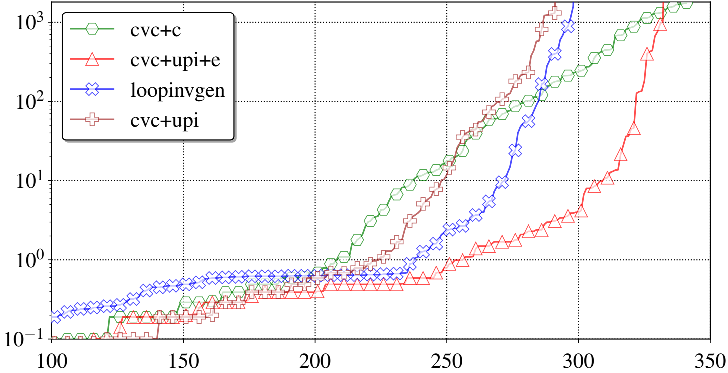
Summary

	Solved	Unique	Total time	Fastest	Shortest
CVC+C	**341**	30	436251s	245	259
CVC+UPI	291	3	494534s	236	231
LOOPINVGEN	298	7	433273s	261	**289**



Summary

	Solved	Unique	Total time	Fastest	Shortest
CVC+C	**341**	30	436251s	245	259
CVC+UPI+E	332	**47**	**414356s**	**306**	222
CVC+UPI	291	3	494534s	236	231
LOOPINVGEN	298	7	433273s	261	**289**
CVC-PORT	400	-	31476s	379	306



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

SyGuS-COMP 2019

Inv Track (829)

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

- 829 benchmarks from the literature in loop invariant synthesis
- 3600s timeout

Injecting some welcome realism

- Kind 2 employs in cooperation:
  - IC3 [Bradley 2011]
  - k-induction [Sheeran et al. 2000]
  - Generation of auxiliary invariants [Kahsai et al. 2011]
- Kind 2 solves all the 480 benchmarks it its test suite in less than 120s
- Considering k-induction in isolation, CVC-PORT is competitive

	Solved	Unique	Time (commonly solved)
CVC-PORT	**323**	**82**	109.6
Kind 2 (k-induction)	313	72	**9.6**

Injecting some welcome realism

- Kind 2 employs in cooperation:
  - IC3 [Bradley 2011]
  - k-induction [Sheeran et al. 2000]
  - Generation of auxiliary invariants [Kahsai et al. 2011]
- Kind 2 solves all the 480 benchmarks it its test suite in less than 120s
- Considering k-induction in isolation, CVC-PORT is competitive

	Solved	Unique	Time (commonly solved)
CVC-PORT	**323**	**82**	109.6
Kind 2 (k-induction)	313	72	**9.6**

- 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

$$f(0,0,0,1,2,1,0) \diamond f(1,0,0,5,2,1,3)$$

    - Can drastically reduce search space

- Improving sample
  - Reducing noise: make points as similar as possible

$$f(1,0,0,1,2,1,0) \diamond f(1,0,0,5,2,1,0)$$

  - 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

FMCAD 2019

2019-10-25, San Jose, CA, USA

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:
$\forall x, y : \; z = x + y$

Result is the sum of the inputs

Invariant Synthesis

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

Invariant?

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

Result is the sum of the inputs

Verification:

$$
\begin{array}{lcl}
z = x \wedge i = 0 \wedge y > 0 & \rightarrow & Inv(x, y, z, i) \\
Inv(x, y, z, i) \wedge i < y \wedge z' = z + 1 \wedge i' = i + 1 & \rightarrow & Inv(x, y, z', i') \\
Inv(x, y, z, i) \wedge i \geq y & \rightarrow & z = x + y
\end{array}
$$

Invariant Synthesis

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

$$Inv(x,\, y,\, z,\, i)$$
$$z = x + i$$
$$z \leq x + y$$

Post-condition:
$\forall x, y : \; z = x + y$

Result is the sum of the inputs

Verification:

$$\begin{array}{lcl} z = x \wedge i = 0 \wedge y > 0 & \rightarrow & Inv(x,\, y,\, z,\, i) \\ Inv(x,\, y,\, z,\, i) \wedge i < y \wedge z' = z + 1 \wedge i' = i + 1 & \rightarrow & Inv(x,\, y,\, z',\, i') \\ Inv(x,\, y,\, z,\, i) \wedge i \geq y & \rightarrow & z = x + y \end{array}$$

Invariant Synthesis in SyGuS

- State-of-the-art: LoopInvGen [Padhi and Millstein 2017]: *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

Alur, Rajeev et al. (2013). "Syntax-guided synthesis". In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, pp. 1–8.

Alur, Rajeev, Arjun Radhakrishna, and Abhishek Udupa (2017). "Scaling Enumerative Program Synthesis via Divide and Conquer". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by Axel Legay and Tiziana Margaria. Vol. 10205. Lecture Notes in Computer Science, pp. 319–336.

Bradley, Aaron R. (2011). "SAT-Based Model Checking without Unrolling". In: Verification, Model Checking, and Abstract Interpretation (VMCAI). Ed. by Ranjit Jhala and David Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 70–87.

Kahsai, Temesghen, Yeting Ge, and Cesare Tinelli (2011). "Instantiation-Based Invariant Discovery". In: NASA Formal Methods. Ed. by Mihaela Gheorghiu Bobaru et al. Vol. 6617. Lecture Notes in Computer Science. Springer, pp. 192–206.

Neider, Daniel, Shambwaditya Saha, and P. Madhusudan (2018). "Compositional Synthesis of Piece-Wise Functions by Learning Classifiers". In: ACM Trans. Comput. Log. 19.2, 10:1–10:23.

Padhi, Saswat and Todd D. Millstein (2017). "Data-Driven Loop Invariant Inference with Automatic Feature Synthesis". In: CoRR abs/1707.02029. arXiv: 1707.02029.

References

Reynolds, Andrew et al. (2019). "cvc4sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis". In: Computer Aided Verification (CAV), Part II. Ed. by Isil Dillig and Serdar Tasiran. Vol. 11562. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 74–83.

Sheeran, Mary, Satnam Singh, and Gunnar Stålmarck (2000). "Checking Safety Properties Using Induction and a SAT-Solver". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Warren A. Hunt Jr. and Steven D. Johnson. Vol. 1954. Lecture Notes in Computer Science. Springer, pp. 108–125.

Solar-Lezama, Armando et al. (2006). "Combinatorial sketching for finite programs". In: Architectural Support for Programming Languages and Operating Systems (ASPLOS). Ed. by John Paul Shen and Margaret Martonosi. ACM, pp. 404–415.

Udupa, Abhishek et al. (2013). "TRANSIT: specifying protocols with concolic snippets". In: Conference on Programming Language Design and Implementation (PLDI). Ed. by Hans-Juergen Boehm and Cormac Flanagan. ACM, pp. 287–296.