DryadSynth : A Concolic SyGuS Solver

Kangjing Huang   Yanjun Wang   Xiaokang Qiu

Department of Electical and Computer Engineering, Purdue University

Background : Syntax-Guided Synthesis (SyGuS)

Program Synthesis

- Problem: Generate program automatically from high-level specifications
  - From WHAT to HOW

Program Synthesis

- Problem: Generate program automatically from high-level specifications
  - From WHAT to HOW

Program Synthesis

- Problem: Generate program automatically from high-level specifications
  - From WHAT to HOW

Specification → Synthesizer → Program

- Combining high-level insights and low-level implementations
  - Low-level is natural for computers
  - High-level is a more humanly job
- Central Challenge: Establishing a synergy between the programmer and the synthesizer

Formal Expression of an Example Synthesis Problem

- Problem: Synthesis of the `max2` function, which accepts 2 arguments and returns the larger-in-value one.
  - Problem must be defined on a certain background theory (eg. LIA, Linear Integer Arithmetics)

Formal Expression of an Example Synthesis Problem

- Problem: Synthesis of the `max2` function, which accepts 2 arguments and returns the larger-in-value one.
  - Problem must be defined on a certain background theory (eg. LIA, Linear Integer Arithmetics)

- Find an LIA binary function $f(x, y)$ such that

$$(f(x,y) = x \vee f(x,y) = y) \wedge f(x,y) \geq x \wedge f(x,y) \geq y$$

Formal Expression of an Example Synthesis Problem

- Problem: Synthesis of the `max2` function, which accepts 2 arguments and returns the larger-in-value one.
  - Problem must be defined on a certain background theory (eg. LIA, Linear Integer Arithmetics)
- Find an LIA binary function $f(x, y)$ such that

$$(f(x, y) = x \lor f(x, y) = y) \land f(x, y) \geq x \land f(x, y) \geq y$$

- Find a program that meets a correctness specification ("constraints") given as a logical formula.

Formal Expression of an Example Synthesis Problem

- Problem: Synthesis of the `max2` function, which accepts 2 arguments and returns the larger-in-value one.
  - Problem must be defined on a certain background theory (eg. LIA, Linear Integer Arithmetics)
- Find an LIA binary function $f(x, y)$ such that

$$(f(x,y) = x \lor f(x,y) = y) \land f(x,y) \geq x \land f(x,y) \geq y$$

- Find a program that meets a correctness specification ("constraints") given as a logical formula.
- Solution `f(x, y) = ite(x >= y, x, y)`

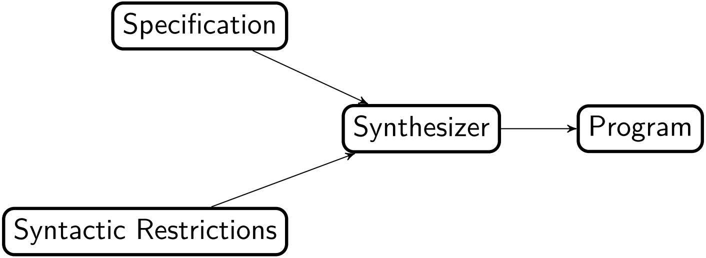
Syntax-Guided Synthesis: Concept [1]

- Supplement the logical specification with a syntactic template
  - The space of allowed implementation is constrained
- Benefits
  - Constrained space make the problem more tractable
  - Specified syntactic constraints could be used in optimizations
- Essentially combines humanly insight into computer's low-level rapidness

[1] Syntax-Guided Synthesis, R. Alur, et al. In 13th International Conference on Formal Methods in Computer-Aided Design, 2013.

Syntax-Guided Synthesis: Concept [1]

- Supplement the logical specification with a syntactic template
  - The space of allowed implementation is constrained
- Benefits
  - Constrained space make the problem more tractable
  - Specified syntactic constraints could be used in optimizations
- Essentially combines humanly insight into computer's low-level rapidness



[1] Syntax-Guided Synthesis, R. Alur, et al. In 13th International Conference on Formal Methods in Computer-Aided Design, 2013.

max2 in SyGuS [2]

```
(synth-fun max2 ((x Int) (y Int)) Int
((Start Int (x y 0 1
             (+ Start Start)
             (- Start Start)
             (ite StartBool Start Start)))

 (StartBool Bool ((and StartBool StartBool)
                  (or StartBool StartBool)
                  (not StartBool)
                  (<= Start Start)
                  (= Start Start)
                  (>= Start Start)))))
```

- Uses a context-free grammar to set the syntactic restrictions on the problem.

[2] Described in the SyGuS-IF language, arXiv:1405.5590v2 [cs.PL] 23 Oct 2016

SyGus Problem Description

Essences of a SyGuS problem:

- A Fixed Background theory T: Fixed types and operations
- Function(s) to be synthesized: name(s) f with type(s)
- Inputs to the problem:
  - Specification ϕ as typed formula using symbols in T and symbol f
  - Context-free grammar G, characterizing the set of allowed expressions $[[G]]$ in T
- Find expression e in $[[G]]$ such that $\phi[f/e]$ is valid in T

SyGuS-COMP: Tracks

- SyGuS-COMP: A series of competitions for solvers on SyGuS problems.
  - Problems and solvers are organized into tracks.
  - Track: A set of problems classified by their predefined background theory and syntactic restrictions.
- SyGuS-COMP'17 has 5 tracks:
  - General Track
  - INV Track: Invariant Synthesis Track
  - CLIA Track: Conditional Linear Integer Arithmetics Track
  - PBE Strings Track: Program By Examples on Strings Track
  - PBE Bitvectiors Track: Program By Examples on Bitvectors Track

SyGuS-COMP: Tracks

- SyGuS-COMP: A series of competitions for solvers on SyGuS problems.
  - Problems and solvers are organized into tracks.
  - Track: A set of problems classified by their predefined background theory and syntactic restrictions.
- SyGuS-COMP'17 has 5 tracks:
  - General Track
  - INV Track: Invariant Synthesis Track
  - CLIA Track: Conditional Linear Integer Arithmetics Track
  - PBE Strings Track: Program By Examples on Strings Track
  - PBE Bitvectiors Track: Program By Examples on Bitvectors Track
- DryadSynth took part in SyGuS-COMP'17 and participated in CLIA and INV track.

CLIA Track

- Theory: Linear Integer Arithmetics
  - Only linear operations are allowed in the expressions
  - All variable types are limited to integers only
- Syntactic Restrictions: Conditional LIA functions
  - Only operations in theory and conditional expressions are allowed in function expression.
  - Only linear operations and ITEs (if-then-else) are allowed.

INV Track

- Theory: LIA
- Syntactic Restrictions: Conditional LIA predicates
  - Same as CLIA, except expressions should be predicates rather than functions..

INV Track

- Theory: LIA
- Syntactic Restrictions: Conditional LIA predicates
  - Same as CLIA, except expressions should be predicates rather than functions..

- All specifications are structured in form of pre-condition, post-condition and a transition relation

INV Track Problem Description

Given predicate on certain integer varaibles and their primed versions:

$$\mathrm{Pref}(x, y), \mathrm{Transf}(x, y, x', y'), \mathrm{Postf}(x, y)$$

Find a Conditional LIA predicate $\mathrm{Invf}(x, y)$ such that

$$\mathrm{Pref}(x, y) \Rightarrow \mathrm{Invf}(x, y)$$

$$\mathrm{Transf}(x, y, x', y') \wedge \mathrm{Invf}(x, y) \Rightarrow \mathrm{Invf}(x', y')$$

$$\mathrm{Invf}(x, y) \Rightarrow \mathrm{Postf}(x, y)$$

INV Track Problem Description

Given predicate on certain integer varaibles and their primed versions:

$$\mathrm{Pref}(x, y), \mathrm{Transf}(x, y, x', y'), \mathrm{Postf}(x, y)$$

Find a Conditional LIA predicate $\mathrm{Invf}(x, y)$ such that

$$\mathrm{Pref}(x, y) \Rightarrow \mathrm{Invf}(x, y)$$

$$\mathrm{Transf}(x, y, x', y') \wedge \mathrm{Invf}(x, y) \Rightarrow \mathrm{Invf}(x', y')$$

$$\mathrm{Invf}(x, y) \Rightarrow \mathrm{Postf}(x, y)$$

- Essentially catches the invariant in a loop designated by the input predicates.
- In our algorithm, INV and CLIA are actually similar problems.

DryadSynth: Approach and Optimizations

DryadSynth

- Explicit + Symbolic Search
- Decision-tree Representation
- One Solver for 2 tracks: CLIA + INV
- Lightweight Implementation based on Z3 (LoC $<$ 2k)

CLIA Functions: Decision Tree Representation

- Consider n-ary CLIA function $f(x_1, \ldots, x_n)$
- Observe that every atomic LIA (in)equation could be converted to form $p(c_0, c_1, \ldots, d_n) \geq 0$, with p being a normalized linear expression:

$$p(c_0, c_1, \ldots, c_n) = c_0 + \sum_{i=1}^{n} c_i x_i$$

CLIA Functions: Decision Tree Representation

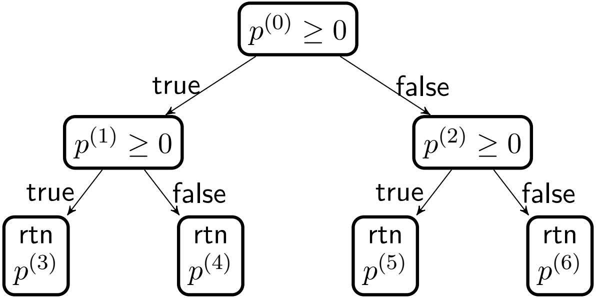
- Consider n-ary CLIA function $f(x_1, \ldots, x_n)$
- Observe that every atomic LIA (in)equation could be converted to form $p(c_0, c_1, \ldots, d_n) \geq 0$, with p being a normalized linear expression:

$$p(c_0, c_1, \ldots, c_n) = c_0 + \sum_{i=1}^{n} c_i x_i$$

- Any CLIA function could be normalized to a binary tree of normalized linear expressions
  - Non-leaf nodes are ITE expressions, with $p \geq 0$ being condition predicate
  - Leaf nodes are return expressions, returning p as function value

Denote that normalized linear expression

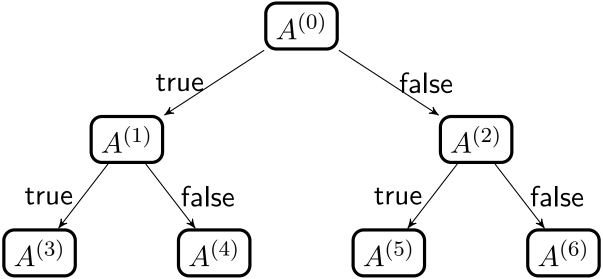
$$p^{(i)} = p(c_0^{(i)}, c_1^{(i)}, \ldots, c_n^{(i)})$$



Denote that the coefficient array in $p^{(i)}$ as

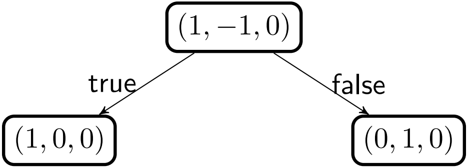
$$A^{(i)} = (c_0^{(i)}, c_1^{(i)}, \ldots, c_n^{(i)})$$

The decision tree could be stored in form



Example: `max2` decision tree

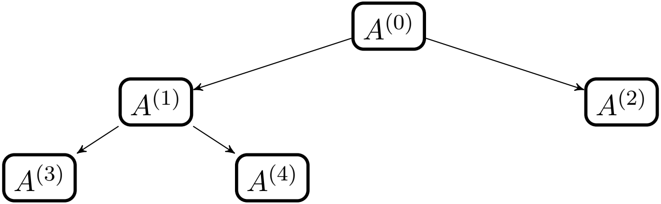
$$(a, b, c) \to ax + by + c$$

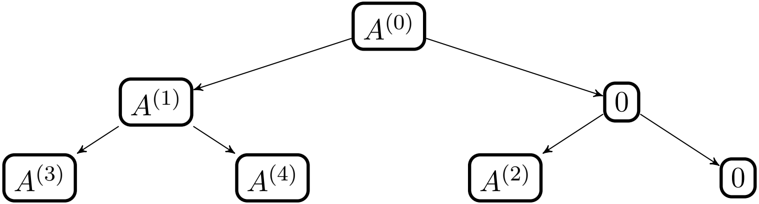



```
max2(x, y) = ite(x - y >=0 , x, y)
```

Full decision trees

An non-full decision tree could always find a full decision tree equivalent.

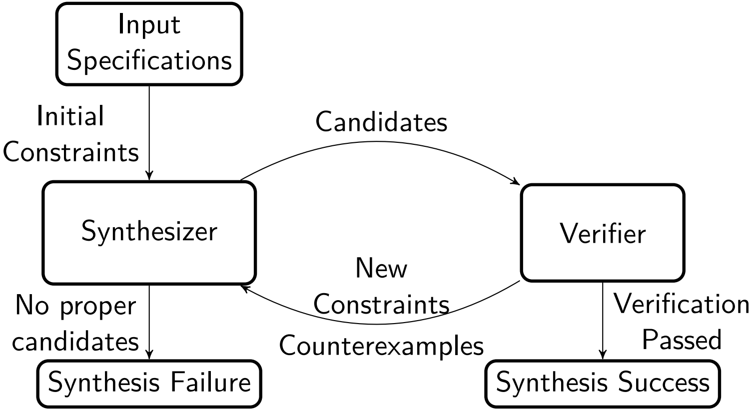




- Completeness for CLIA functions is guaranteed.

CEGIS Framework

- CEGIS: CounterExample Guided Inductive Synthesis [3]
- Candidate-Counterexample iterations drive inductive synthesis.



[3] A. Solar-Lezama, et al. "Combinatorial sketching for finite programs," in ASPLOS'06. ACM, 2006, pp. 404–415.

DryadSynth: Symbolic search in CEGIS

- Symbolic search for fixed tree height is performed in CEGIS loop.
- For a fixed height h, decision tree represents a function with non-concrete coefficients

$$f(x_1, \dots, x_n) = ite(p_0 \geq 0, ite(\dots), \dots)$$

- A concrete input point makes it a function of coefficients

$$f(w) = g(c_0^{(0)}, c_1^{(0)}, \dots)$$

W is a concrete point of the variables.

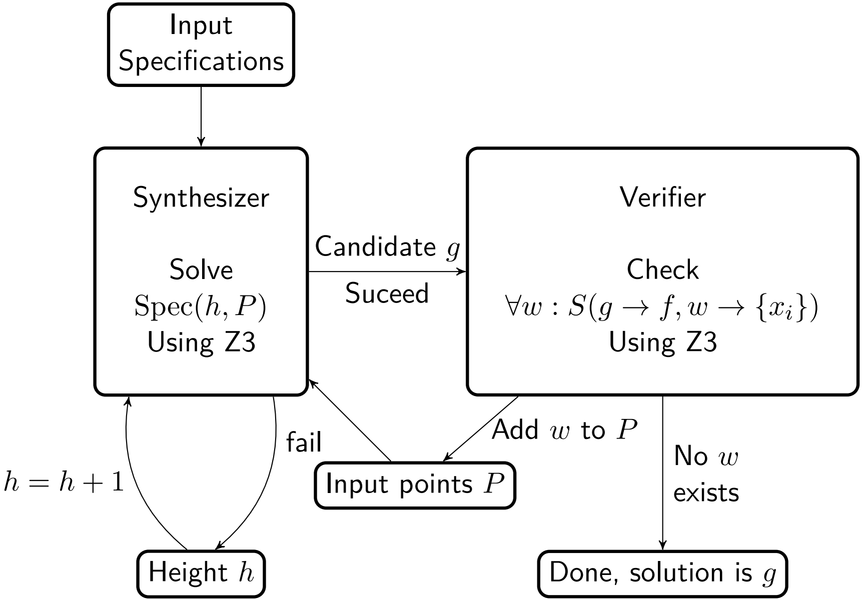
DryadSynth: Enumerative search in heights

- Concrete expression could be substituted back to constraint, make constraints effectively constraints on coefficients. So for a set of input points P.

$$\mathrm{Spec}(h, P) = \bigwedge_{w \in P} S(f(w) \to f, w \to (x_1, \ldots, x_n))$$

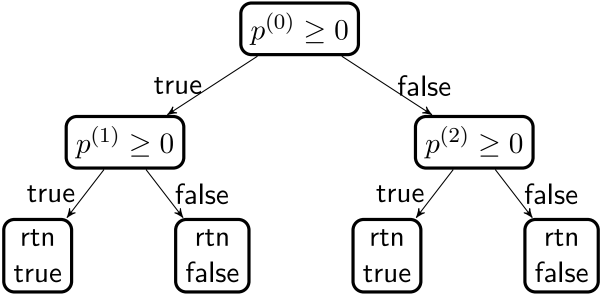
- Search from $h = 1$, enumeratively increase h if solution not found on current height.
- Checking and Solving is done with SMT Solver Z3 [4]

[4] L. de Moura and N. Bjørner, Z3: An Efficient SMT Solver. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340


Input Specifications
Synthesizer
Solve $\mathrm{Spec}(h, P)$ Using Z3
Candidate g
Suceed
Verifier
Check $\forall w : S(g \to f, w \to \{x_i\})$ Using Z3
fail
$h = h + 1$
Add w to P
No w exists
Input points P
Height h
Done, solution is g

DryadSynth on INV problems

- INV problems are effectively CLIA predicate synthesis problems with restrictions on the form of the specifications.
- A CLIA predicate could be expressed in decision tree format as:



- Thus our previous approach could be easily adapted.

Optimizations on DryadSynth

- Parallelization
  - Naturally, CEGIS loop on different heights shall be independent to each other
  - Parallelization could leverage multi-cores
- Coefficient Bounds
  - Certain bounds set on coefficient could do a performance boost to the algorithm
- Prescreen Analysis
  - Prescreen of input problem could get rid of certain trivial problems that have a performance impact on the algorithm.

Parallelization

- Different h has different CEGIS processes
  - Counterexample sets do not need to be shared
  - Different CEGIS processes are entirely independent.
- On a n-core machine
  - Run on $h = 1 \to n$ initially
  - Set up a timeout for each thread
  - When a thread times out or yields no solution, process to next height that has not been processed yet.

Coefficient Bounds

- Observations on typical CLIA problems indicate that
  - Most of the solutions' coefficients are very small, with lots of the coefficients being ± 1
  - In rare cases, when solution contain large coefficient, there're often large coefficients in specifications
  - With coefficient bounded by certain bounds, a significant improvement in Z3 performance cound be archieved
    - This is due to the undeterministic nature of Z3's algorithm.

- In DryadSynth, we thus set Coefficient bounds for coefficients in synthesis
  - This effectively adds contraint in form of $a \leq c_i^{(j)} \leq b$ to $\mathrm{Spec}(h, P)$
  - DryadSynth split the coefficient search region into 3 parts
    - $0 \leq |c| \leq 1$
    - $1 < |c| \leq c_b$
    - $|c| > c_b$
  - When a CEGIS process on lower region times out or yields no solution, advance to next region.
  - Set of counterexamples need to be shared between regions
  - c_b is a bound chosen manually

Prescreen Analysis

- Some trival cases have a huge impact on the naive approach's performance
- Unused variables: variables that defined and used in sythesis target parameters, but could be logically guaranteed that would not appear in solution
  - Z3's performance drops significantly when expression size increases
  - Eliminating unused variables would improve such performance
  - Archieved by logical analysis of input specifications.

Results and Future Improvements

Results

- In SyGuS-COMP'17:

Results

- In SyGuS-COMP'17:
- On CLIA tracks, total of 73 problems
  - DryadSynth solved 32 in time.
  - Winner solved all in time.
- On INV tracks, total of 74 problems
  - DryadSynth solved 64 in time.
  - Winner solved 65 in time.
- Not so good performance in CLIA, but pretty good in INV.

Analysis

- Many CLIA problems are deep-in-height in nature
  - `max_n` and `array_search_n`
  - Deep decision trees are disasters to Z3 performance
  - But they're simple in problem formations
    - Could use other approaches to solve rather than CEGIS
- Most INV problems are simpler in tree structures, but somehow complex in formation
  - Best for decision tree representation
  - DryadSynth may suit better on INV problems.

Possible Improvements

- Further Parallelization
  - Can different coefficient regions on same height be parallelized?
  - Possible, but need to take care of the counterexamples
- Adaptive coefficient bounds
  - As indicated before, coefficient value depends on coefficients in specifications
  - Set up coefficient bounds according to specification
- Other approaches
  - Idea: apply further syntactic constraints through assumptions that could capture a meaningful number of problems
  - Example: Single Invocation Problems.