Introduction to program synthesis

Tim Blazytko
⟨tim.blazytko@rub.de⟩

Ruhr-Universität Bochum

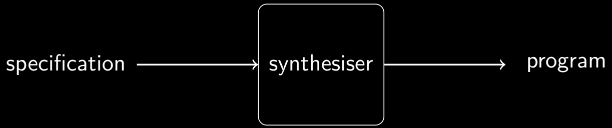
24th February 2017

Today

- What is program synthesis?
- How does it work?
- What is the state-of-the-art?
- How can we use it for binary program analysis?

Note: This talk mainly summarises existing work.

Program synthesis
Definition



Program synthesis

Automatic construction of programs that satisfy a given specificaton.

Motivation

Expression simplification

Given the following two functions:

$$g(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$h(x, y, z) := x + y + z$$

We can prove that

$$g(x, y, z) = h(x, y, z)$$

We want to learn the function h.

Building blocks

We have to specify three categories

Program specificaton

How do we specify the intended program behaviour?

Synthesis language

In which language do we synthesise our program?

Synthesis method

How do we synthesise our program?

We will explore different combinations.

Specifying program behaviour

Logical specificaton

$$\phi_{\texttt{spec}}(\vec{I}, O) := O = g(x, y, z)$$

- input vector $\vec{I} := (x, y, z)$
- ouput $O := g(x, y, z)$
- conrete formula that specifies the input-output mapping

Enumerative program synthesis

Algorithm

Enumerative program synthesis

1. perform an exhaustive search
2. sieve empirically/filter candidates
3. check for semantic equivalence to specification

Enumerative program synthesis

Example

- generate set of candidates with exhaustive search
- filter set of candidates heuristically (not further specified)
- $x + x + x$ and $x + y + z$ are (remaining) hypothesis candidates
- check each candidate for semantic equivalence to the specificaton
- $x + y + z$ is (proven) semantically equivalent to specification

Component-based program synthesis

Logical encoding

Logical specifications

- program behaviour
- set of library components
- data-flow between components
- encoding of a well-formed program

You don't wanna see these encodings here.

Synthesis result

- permutation of library components and input-output relations

Component-based program synthesis

Example

We specify the components:

- $f_a(i_0, i_1, O) := O = i_0 + i_1$
- $f_b(i_0, i_1, O) := O = i_0 + i_1$

Synthesised program $h(x, y, z)$

Input: x, y, z

Output: O

1. $O_1 := f_b(x, y)$
2. $O_2 := f_a(O_1, z)$
3. $O := O_2$

- $O_1 = x + y$
- $O_2 = (x + y) + z$

Oracle-guided program synthesis

Input/output samples as program specification

$$\text{ORACLE}(\vec{I}) := g(x, y, z)$$

- access to I/O oracle
- partial program specification based on I/O samples
- finite set of samples: $S = \{(1, 1, 1) \rightarrow 3, (2, 4, 3) \rightarrow 9, \dots\}$

Template-based program synthesis

Problem

$$\exists f : \bigwedge_{\vec{I}, O \in S} f(\vec{I}) = O$$

Does there exist a function for which . . . ?

- quantification over functions is second-order logic
- SMT solvers operate on fragments of fist-order logic

⇒ function templates come to the rescue

Template-based program synthesis

Function templates

We define function template T with free coefficients $\vec{c} = (c_0, \dots, c_{n-1})$

$$T(c_0, c_1, x, y, z) := (c_0 == 0)?(x + y + c_1) : (x + c_1 + c_1)$$

$$c_1 \in \{x, y, z\}$$

$$\varphi := \exists \vec{c} : \bigwedge_{\vec{I}, O \in S} T(\vec{c}, \vec{I}) = O$$

Does there exist an assignment for which ...?

$\Rightarrow$ SMT(φ) returns $c_0 = 0$ and $c_1 = z$

Counter-example guided program synthesis

Learning semantics incrementally

- use logical specification as I/O oracle
- use SMT solver to obtain *distinguishing* inputs

Algorithm

1. query I/O oracle with input vector
2. search a program candidate φ that satisfies I/O behaviour
3. check semantically equivalence to specification
4. if not equivalent
    1. obtain distinguishing input and goto 1
5. return program candidate

Counter-example guided program synthesis

Example

- $S := \{(1,1,0) \to 2\}$
- possible program candidate: $h(x,y,z) := x + y$
- $\text{SMT}(h(x,y,z) \neq g(x,y,z)) \in \text{SAT}$

$\Rightarrow$ counterexample: $(1,2,3)$

- query $\text{ORACLE}(1,2,3) = 6$
- $S := \{(1,1,0) \to 2, (1,2,3) \to 6\}$
- possible program candidate: $h(x,y,z) := x + y + z$
- $\text{SMT}(h(x,y,z) \neq g(x,y,z)) \in \text{UNSAT}$
- return h

Stochastic program synthesis

A new era

- stochastic optimisation problem
- approximate global optima

⇒ intermediate results instead of SAT/UNSAT

- synthesis is guided by a cost function towards global optima
- one example: Monte Carlo Markov Chains (MCMC)
- next talk introduces another approach in detail

Stratified program synthesis

Learn more complex programs iteratively

Algorithm

1. synthesise expressions
2. add synthesised expressions to synthesis language
3. goto 1

Learn more complex expressions from previous results.

Stratified synthesis

Example

- the synthesis language's components are $+$, a and b
- we want to synthesise $a + a + a + a + b + b$
- we synthesise $a + b$
- we extend the synthesis language: $+$, a, b and $a + b$
- we synthesise $(a + b) + (a + b) + a + a$

Applications

Stochastic superoptimization (STOKE)

- find an optimal code sequence for a sequence of instructions
- replace assembly code by equivalent faster code
- stochastic cost minimisation problem
- cost function for transformation correctness and performance improvements
- MCMC for search space exploration
- shorter and faster programs than `gcc -O3`

Applications

Learning processor instructions from I/O samples

- I/O samples of CPU instructions
- template-based synthesis approach
- 6 synthesis templates for different ALU instructions
- learned over 500 Intel x86 instructions in less than two hours

Automated generation of intermediate representations for program analysis

Applications

Learning formal semantics for Intel x86

- a manually written base set of formal semantics for 51 instructions
- stratified synthesis with STOKE as synthesis core
- learned formal semantics for $1,795.42$ instructions (61.5% of the instructions in scope)
- found errors in Intel documentation

Applications

Metamorphic extraction

- obfuscated metamorphic code engine
- mixture of template-based and counter-example guided approach
- I/O pairs from obfuscated metamorphic engine
- template of metamorphic engine
- SMT solver guessed assignment
- terminate if synthesised and obfuscated engines are semantically equivalent

Applications

Shellcode generation

- SMT-based approach
- shellcode functionality
- shellcode encoding restrictions
  - null bytes
  - all bytes must be odd
  - only prime bytes

Applications

Deobfuscation

- learning semantics of obfuscated codes
- simplifying obfuscated code
- see next talk for details

Limitations

- operate on semantic complexity
- non-deterministic functions
- point functions
- confusion and diffusion (cryptography)

Conclusion

- specifying program behaviour
- enumerative program synthesis
- component-based program synthesis
- oracle-guided program synthesis
- template-based program synthesis
- counter-example guided program synthesis
- stochastic program synthesis
- stratified program synthesis
- superoptimisation
- CPU emulator synthesis
- shellcode generation
- deobfuscation

References I

- Sorav Bansal and Alex Aiken. 'Automatic Generation of Peephole Superoptimizers'. In: *ACM Sigplan Notices*. 2006.
- Patrice Godefroid and Ankur Taly. 'Automated Synthesis of Symbolic Instruction Encodings from I/O Samples'. In: *ACM SIGPLAN Notices*. 2012.
- Sumit Gulwani. 'Dimensions in Program Synthesis'. In: *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming*. 2010.
- Sumit Gulwani et al. 'Synthesis of Loop-free Programs'. In: *ACM SIGPLAN Notices* (2011).
- Stefan Heule et al. 'Stratified synthesis: Automatically Learning the x86-64 Instruction Set'. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 2016.

References II

- Susmit Jha et al. 'Oracle-guided Component-based Program Synthesis'. In: *ACM/IEEE 32nd International Conference on Software Engineering*. 2010.
- Rolf Rolles. *Program Synthesis in Reverse Engineering*. http://www.msreverseengineering.com/blog/2014/12/12/program-synthesis-in-reverse-engineering. 2014.
- Rolf Rolles. *Synesthesia: A Modern Approach to Shellcode Generation*. http://www.msreverseengineering.com/blog/2016/11/8/synesthesia-modern-shellcode-synthesis-ekoparty-2016-talk. 2016.
- Eric Schkufza, Rahul Sharma and Alex Aiken. 'Stochastic Superoptimization'. In: *ACM SIGPLAN Notices* (2013).