Synthesis
Reading Reflection

Discuss in groups

• So far, do you find enumerative search (tracking the subset of the program space explored so far) more natural? Or symbolic search (representing the space of all valid programs)?

• How are you feeling about viewing programs as manipulable objects rather than text?
  • Have you thought before about the fact that programs themselves are also data and can be treated as inputs to other programs?
Reading Key Takeaways

• Inductive reasoning makes broad generalizations from specific observations
  • \[\rightarrow\] inductive synthesis is about generalizing from ambiguous specifications
• The difference between finding a program that satisfies the spec and finding the program the user actually wants
• A nice review of ASTs and the importance of the DSL design for the program space size (although we touched on these in last class)
• A friendly introduction to symbolic search (representing the space of all valid programs instead of tracking the subset of programs explored so far)
Enumerative → Symbolic (Constraint-Based)
About Rosette

Rosette is a solver-aided programming language that extends Racket with language constructs for program synthesis, verification, and more. To verify or synthesize code, Rosette compiles it to logical constraints solved with off-the-shelf SMT solvers. By combining virtualized access to solvers with Racket's metaprogramming, Rosette makes it easy to develop synthesis and verification tools for new languages. You simply write an interpreter for your language in Rosette, and you get the tools for free!

```racket
#lang rosette

(define (interpret formula)
  (match formula
    [\(^\_\_expr \_\_\) (apply \_\_\_ expr)]
    [\(\_\_\_expr \_\_\_\) (apply \_\_\_ \_\_\_ expr)]
    [\(\_\_\_\_expr \_\_\_\_\_) (! (interpret expr))]
    [lit (constant lit boolean?)])
)```
If you want to get really into Rosette, I recommend...

• https://courses.cs.washington.edu/courses/cse507/19au/index.html
Here be dragons
Let’s tour Rosetteland!

• Can you run this program in DrRacket? Please try to help each other debug if you can’t!

• 

```racket
#lang rosette/safe
(require rosette/lib/synthax)
(current-bitwidth #f)
```
(1) individual Rosette intro activity
(2) group Rosette activity
What did you learn from the Rosette activity?
A few learning goals

You might have learned...

• That you can write a synthesizer!
• That there are many possible ways of designing the grammar, many possible ways of designing the spec
• A visceral understanding of the difference between finding a program that meets your spec and the program you actually want. :) Especially in example-based specs.
• The limits of what you can control in Rosette.

https://people.csail.mit.edu/asolar/SynthesisCourse/Lecture2.htm
Armando Solar-Lezama
One (of many) solutions

(define-synthetic (is-title x depth)
  #:base (choose #t #f)
  #:else (choose
    #t #f
    (if ((choose < >) ((choose get-font-size get-num-words) x) (??))
      (is-title x (- depth 1))
      (is-title x (- depth 1)))))

(define (is-title-synthesized x)
  (is-title x 1))

(define-symbolic i integer?)
(print-forms
  (synthesize
    #:forall (list i)
    #:guarantee (assert (or
      (< i 0)
      (>= i (length texts))
      (equal? (is-title-synthesized (list-ref texts i)) (get-is-title (list-ref texts i)))))))

Welcome to DrRacket, version 7.8 [3m].
Language: rosette/safe, with debugging; memory limit: 256 MB.
20
450
#f
/Users/schasins/Documents/titleDetection.rkt:49:0
'(define (is-title-synthesized x) (if (> (get-font-size x) 31) #t #f))
And this is adaptable as we get more complicated inputs from our user...

Original input-output pairs

Here we add 3 more

The same synthesizer now produces:

```
(define (is-title-synthesized x)
  (if (< (get-font-size x) 69) #f (if (< (get-num-words x) 200) #t #f)))
```
Rosette for more realistic tasks…

Figure 9. The framework sketch $F_{Alg}^{glbe}$ for synthesizing a memory model for the x86 architecture (a), and synthesized models $TSO_0$ and $TSO_4$ before and after resolving ambiguities (b). The expression holes for ppo and grf define a search space of size $2^{254}$, as described in Figure 8. The fences relation is empty because x86 fences are not cumulative.

Abstract

A memory consistency model specifies which writes to shared memory a given read may see. Ambiguities or errors in these specifications can lead to bugs in both compilers and applications. Yet architectures usually define their memory models with prose and litmus tests—small concurrent programs that demonstrate allowed and forbidden outcomes. Recent work has formalized the memory models of common architectures through substantial manual effort, but as new architectures emerge, there is a growing need for tools to aid these efforts.

This paper presents MemSynth, a synthesis-aided system for reasoning about axiomatic specifications of memory models. MemSynth takes as input a set of litmus tests and a framework sketch that defines a class of memory models. The sketch comprises a set of axioms with missing expressions (or holes). Given these inputs, MemSynth synthesizes a completion of the axioms—i.e., a memory model—that gives the desired outcome on all tests. The MemSynth engine
Figure 4. A ToyRISC interpreter using Serval (in Rosette).
Reflections on Rosette

- Concise program -> quite complex and sophisticated synthesizers
- Opacity
- Control
quick prep for next session
To think about for next reading

• You do not need to memorize or deeply understand details of these approaches!
• I want you to recognize the key terms and know where to turn for a high-level overview of key techniques. Also, this chapter offers excellent pointers to examples of synthesis work, which you might find useful if you start tackling a synthesis project.
• Think about
  • How these different approaches would or wouldn’t apply to the synthesis ideas you brainstormed last session
  • How these different approaches shape the user interaction
Install before next class:

Z3 SMT solver

We’ll use the Python Z3 bindings. First make sure you have Python installed. Then install the Z3 bindings. (https://pypi.org/project/z3-solver/)

```
pip install z3-solver
```

OR

```
pip install z3-solver --user
```

Then make sure you can run this program, which I’ll also upload in Slack.

```
from z3 import *

x = Int('x')
y = Int('y')
solve(x > 1, y > 1, x * y + 3 == 7)
```