Programs Synthesis from Refinement Types

Nadia Polikarpova
Ivan Kuraj
Armando Solar-Lezama
(MIT CSAIL)
Program Synthesis

specification

Synthesizer

Explorer

candidate

Verifier

feedback

automatic

modular

executable
Modular Verification for Synthesis
Synthesis of Functional Programs

- input-output examples + testing
  [Escher, Myth, λ²]

- executable assertions + bounded checking
  [STUN, SynthRec]

- refinement types + type checking

- formal specs + deductive verification
  [Leon, Jennysis]
Refinement Types

Nat

max :: x: Nat → y: Nat → \{ v: Nat | x ≤ v ∧ y ≤ v \}

Cons x xs :: \{ v: List Nat \}

data List α where
    Nil :: v: \{ List α | len v = 0 \}
    Cons :: x: α → xs: List α
            → v: \{ List α | len v = len xs + 1 \}

measure len :: List α → Int
    len Nil = 0
    len (Cons _ xs) = len xs + 1

[Rondon et al.’08, Kawaguchi et al.’09]
Example 1: replicate

**measure** len :: List α -> Int

**data** List α where

- Nil :: {v: List α | len v = 0}
- Cons :: x: α -> xs: List α -> {v: List α | len v = len xs + 1}

zero :: {v: Int | v = 0}
inc :: x: Int -> {v: Int | v = x + 1}
dec :: x: Int -> {v: Int | v = x - 1}

**replicate** :: n: Nat -> x: α -> {v: List α | len v = n}
replicate = ??
Example 2: insert

**measure** elems :: BST α -> Set α

**data** BST α **where**

Empty :: {v: BST α | elems v = []}

Node :: x: α ->

  l: BST {α | v < x} ->

  r: BST {α | x < v} ->

  {v: BST α | elems v = elems l + elems r + [x]}

insert :: x: α -> t: BST α -> {v: BST α | elems v = elems t + [x]}

insert = ??
Example 2: insert (solution)

```
insert x t = match t with
    Empty → Node x Empty Empty
    Node y l r → if y = x
      then t
      else if y ≤ x
        then Node y l (insert x r) :: BST { v: α | v > y }
        else Node y (insert x l) r
```
Synthesis from Refinement Types: Take One

Synthesizer

refinement types

Explorer

Liquid Type Inference
[Rondon et al. ’08]

whole-program analysis!

functional program
Synthesis from Refinement Types: Our Solution

![Diagram showing a synthesizer with connections to an explorer, modular refinement type reconstruction, and a functional program. The diagram also includes notes on refinement types and a top-down approach.](image-url)
Modular Verification for Synthesis
Evaluation

- **Lists**
  - take, drop, delete, zip with function, reverse, de-duplicate, fold, length/append with fold, ...

- **Sorting**
  - insertion sort, merge sort, quick sort

- **Binary Search Trees**
  - member, insert, delete

- **User datatypes**
  - AST desugaring

- **Red-black trees**
  - rotation

50 benchmarks

<5 s

19 s
Synquid

bitbucket.org/nadiapolikarpova/synquid