Program Synthesis

Tikhon Jelvis (tikhon@jelv.is)

February 24, 2014

Synthesis

- Find a program to some specification ($\phi(input, output)$)
  - input/output pairs
  - executable specification
- $\exists P \forall x.\phi(x, P(x))$

Why?

- easy
  - spec might be easier than program
- correct
  - verifying spec easier than verifying program

Compilers

- compilers are deterministic
- only consider **correct** optimizations
- hard to write

Synthesizer

- can be non-deterministic
- considers potentially incorrect optimizations
- hard to scale
- accept partial specifications

Programming by Demonstration

- user provides inputs/outputs
- generate program to match
- **interactive**

Example

First	Last	Initials
John	Doe	
Bob	Smith	
Tikhon	Jelvis	
...	...	

Example

First	Last	Initials
John	Doe	J. D.
Bob	Smith	
Tikhon	Jelvis	
...	...	

Example

First	Last	Initials
John	Doe	J. D.
Bob	Smith	B. S.
Tikhon	Jelvis	T. J.
...	...	...

Spec

- **One** input/output:
  - “John” “Doe” $\rightarrow$ “J. D.”
- Usually only a few needed
- Interactive

Small Language

- Targets a custom language
- Heavily limits looping
- Much smaller space of possible programs

Rough Approach

- Uses **Version Space Algebra** (VSA)
- "Space" of possible programs
  - inputs/outputs trim space
- Haskell package: HaVSA

Superoptimization

- optimize existing program
  - synthesize equivalent program
- much easier to implement
- limited scalability

Example

- GreenArrays
  - stack-based architecture
  - uses Forth
  - difficult to use
- Haskell package: array-forth

MCMC

- Markov Chain Monte Carlo (MCMC)
- randomized hill climbing
  - random mutations
  - incorrect code
- Haskell package: mcmc-synthesis (limited)
- needs cluster

SMT

- SMT: SAT modulo theories
- Solve logic formulas
- reasonably fast

Formulas

- compile program to formula
- $\phi(\mathit{input}, \mathit{program}, \mathit{output})$
  - fix *input*, *program*: interpreter
  - fix *program*, *output*: reverse interpreter
    - non-deterministic
  - fix *input*, *output*: synthesis

Verification

- compare two programs **exhaustively**
- problem: $\forall input.spec(input) = program(input)$
- actual: $\exists input.spec(input) \neq program(input)$

CEGIS

- Counter-example guided inductive synthesis (CEGIS)
- solve for program
  - limited set of inputs/outputs
- verify against spec
  - if verified: done
  - else: new input/output pair; repeat
- few pairs needed

Scaling (or not)

- formulas are hard to scale
- **exponential** in program size
- maybe 100 instructions
  - with luck
  - if you're patient

Sketching

- some things are easy for programmers
- some things are easy for solvers
- let programmers write the easy parts!
- specify a program with "holes" and solve for the holes

SBV

- SMT-based verification (sbv)
- Haskell package for SMT solvers
- write symbolic Haskell program
  - prove facts about it
  - solve for variables
    - synthesis!

Credit

- Most of the presentation heavily influenced by:
  - Professor Bodik at the Berkeley ParLab
  - Especially slides from his class on synthesis
- Programming by example:
  - Flash Fill, a team at MSR led by Sumit Gulwani
- GreenArrays:
  - Based on a synthesis project I worked on along with Professor Bodik, Mangpo, Rohin Shah and Nishant Totla
- MCMC-synthesis:
  - Based on a side-project around GreenArrays that I worked on with Jessica Taylor