EECS 219C: Computer-Aided Verification

# Syntax-Guided Synthesis

(selected/adapted slides from FMCAD'13 tutorial by R. Alur)

Sanjit A. Seshia

EECS, UC Berkeley

# Solving SyGuS

- Is SyGuS same as solving SMT formulas with quantifier alternation?

- SyGuS can sometimes be reduced to Quantified-SMT, but not always
  - Set E is all linear expressions over input vars x, y

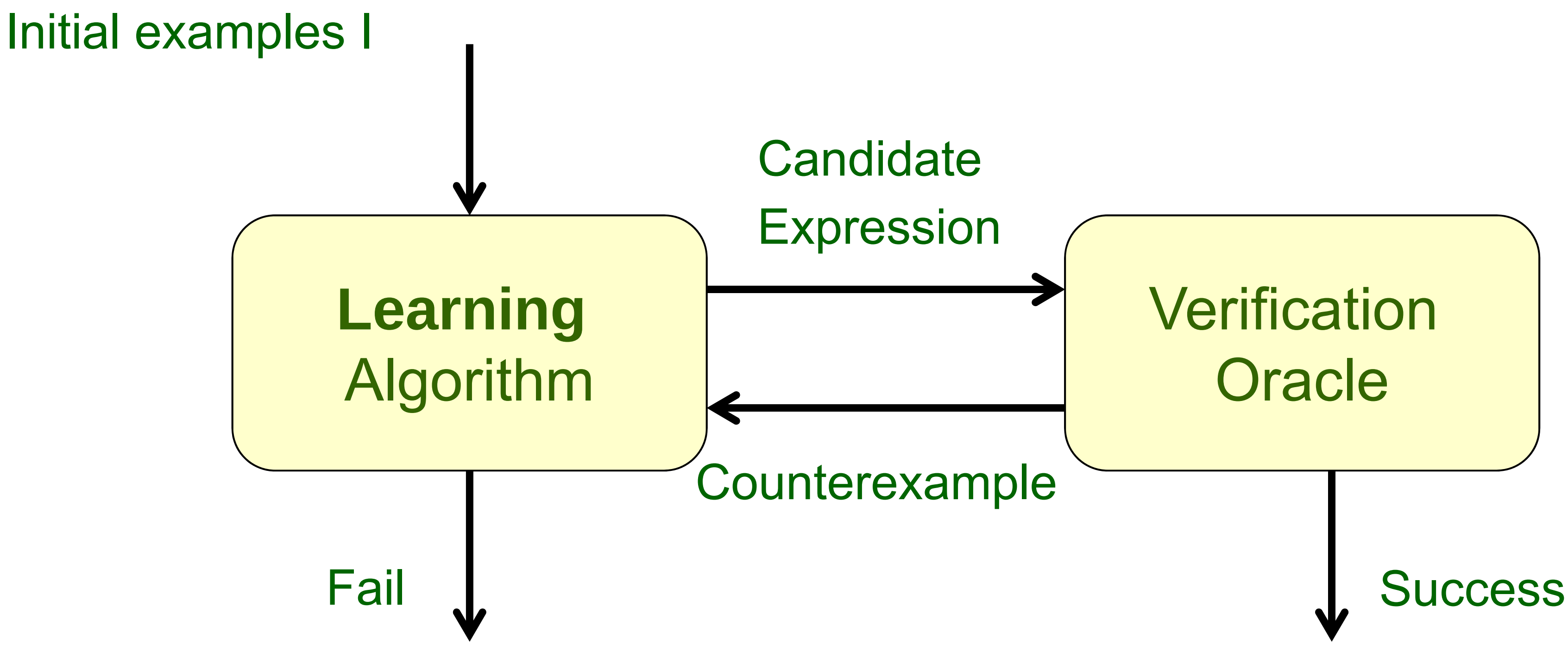
    SyGuS reduces to Exists a,b,c. Forall X. $\varphi$ [ f/ ax+by+c]
  - Set E is all conditional expressions

    SyGuS cannot be reduced to deciding a formula in LIA

- Syntactic structure of the set E of candidate implementations can be used effectively by a solver

- Existing work on solving Quantified-SMT formulas suggests solution strategies for SyGuS

# SyGuS as Active Learning



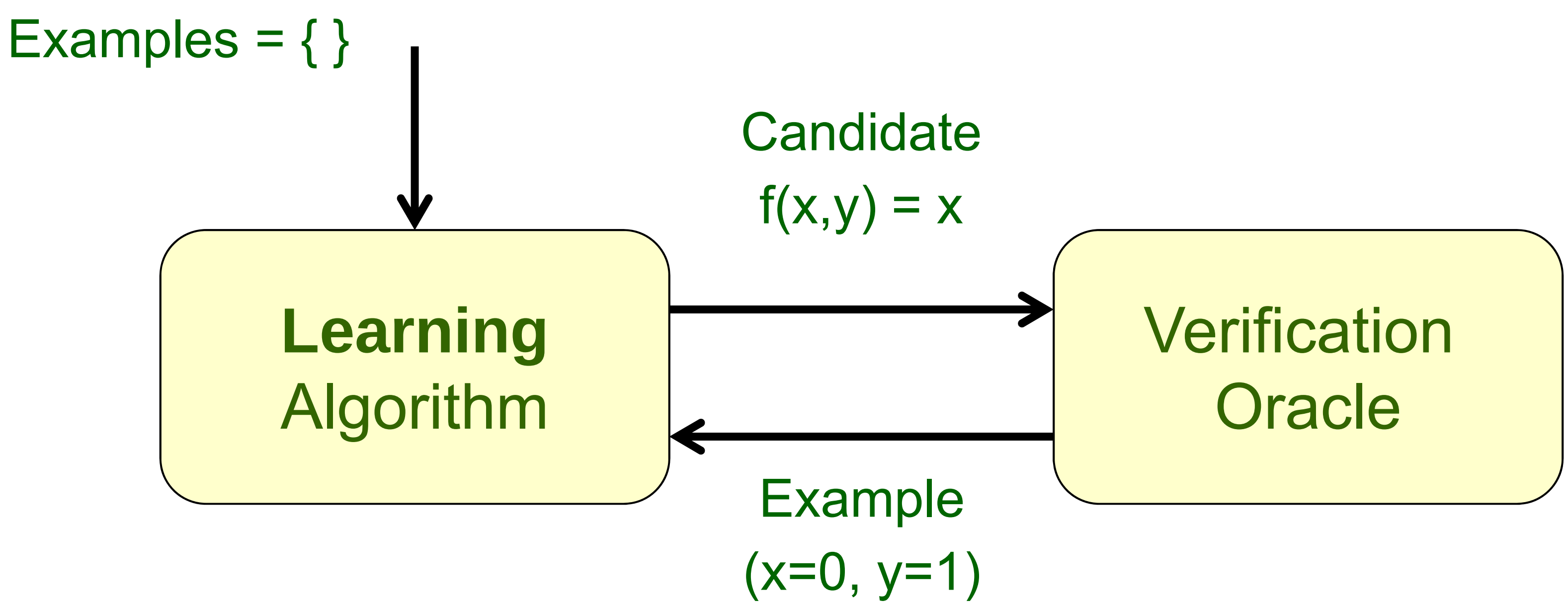

Concept class: Set E of expressions

Examples: Concrete input values

# CEGIS Example

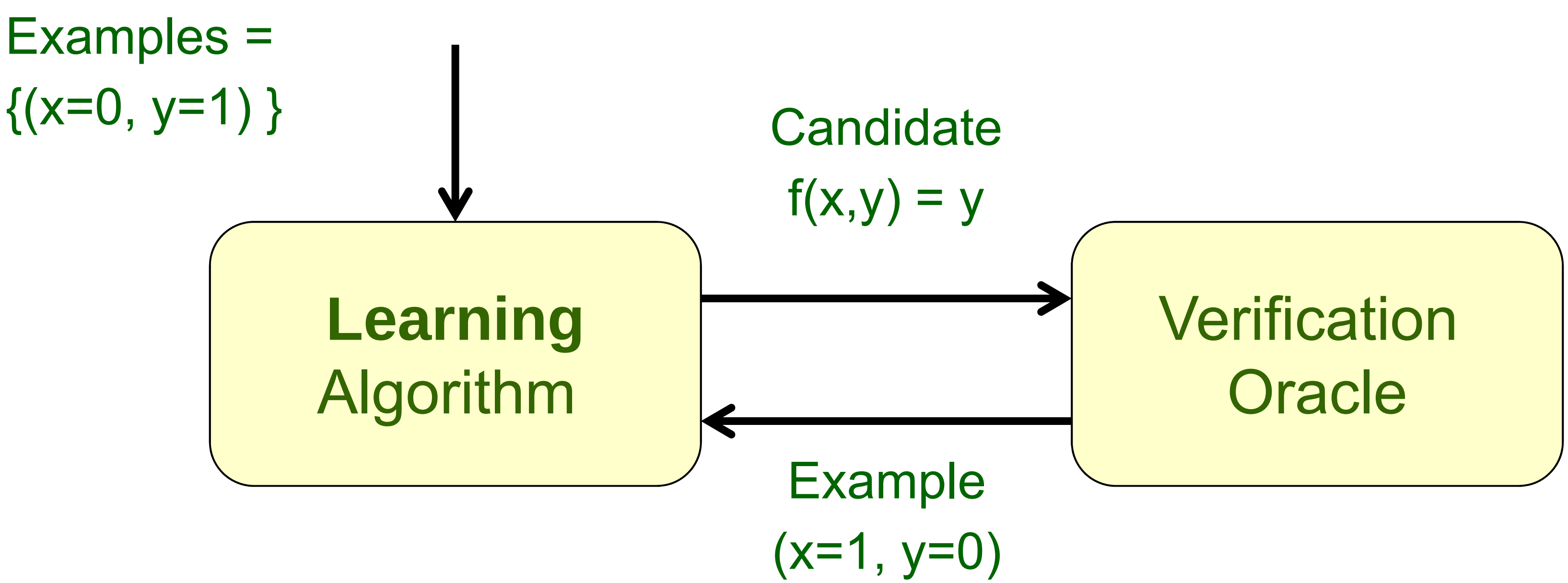
- Specification: $(x \leq f(x,y))$ & $(y \leq f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$

- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



# CEGIS Example

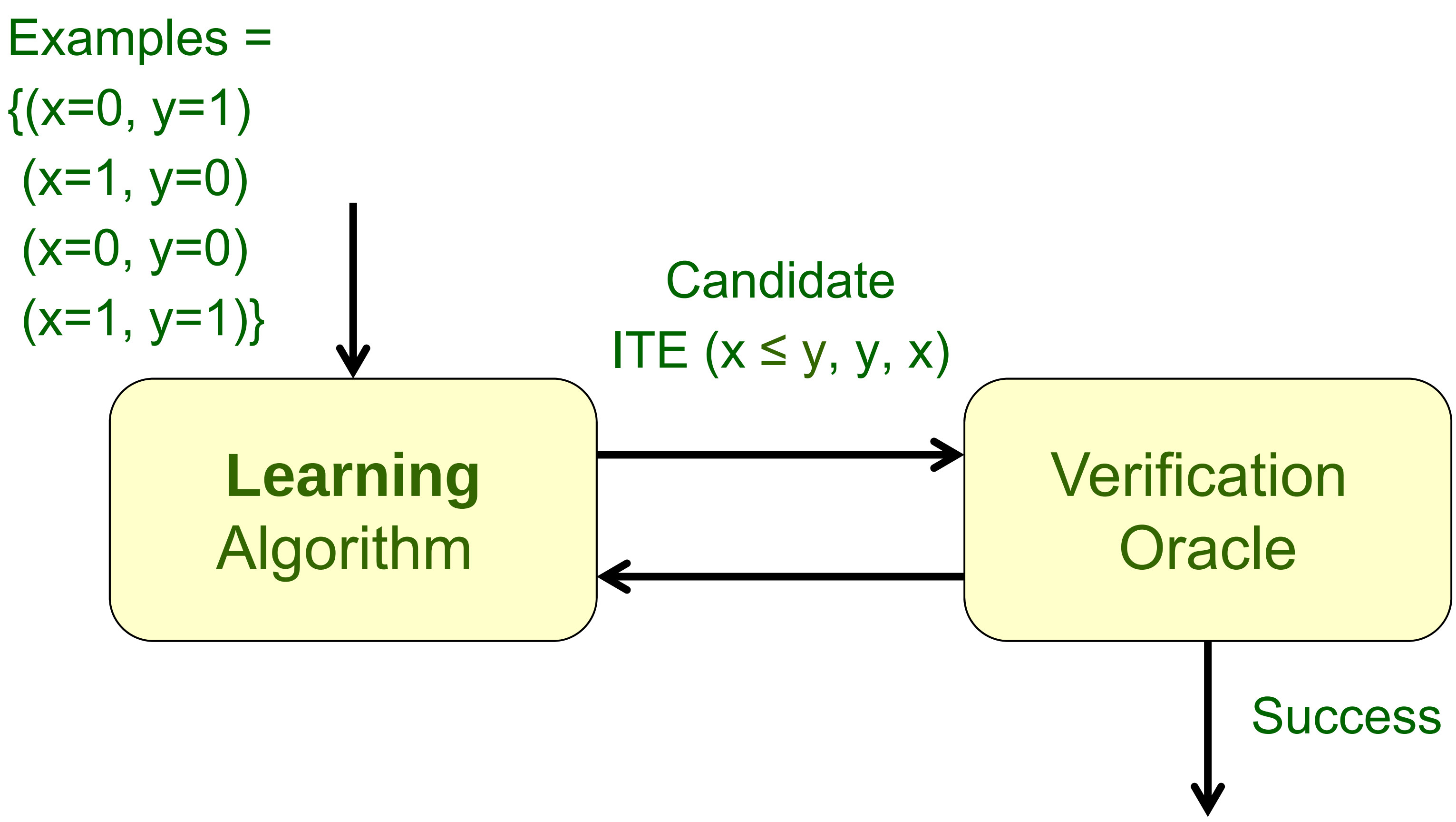
- Specification: $(x \leq f(x,y))$ & $(y \leq f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$
- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



# CEGIS Example

- Specification: (x ≤ f(x,y)) & (y ≤ f(x,y)) & (f(x,y) = x | f(x,y) = y)

- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



# SyGuS Solutions

- CEGIS approach (Solar-Lezama, Seshia et al)

- Coming up: Learning strategies based on:
  - Enumerative (search with pruning): Udupa et al (PLDI'13)
  - Symbolic (solving constraints): Jha et al (ICSE'10,PLDI'11)
  - Stochastic (probabilistic walk): Schkufza et al (ASPLOS'13)

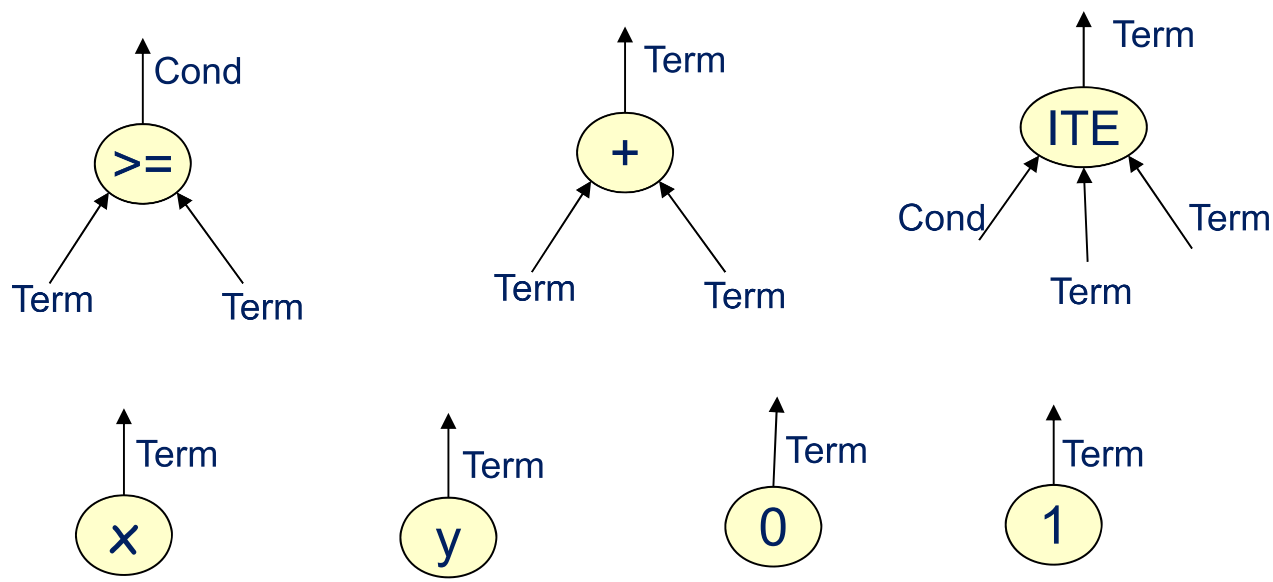
# Enumerative Learning

- Find an expression consistent with a given set of concrete examples
- Enumerate expressions in increasing size, and evaluate each expression on all concrete inputs to check consistency
- Key optimization for efficient pruning of search space:
  - Expressions $e_1$ and $e_2$ are equivalent if $e_1(a,b)=e_2(a,b)$ on all concrete values (x=a,y=b) in Examples
  - (x+y) and (y+x) always considered equivalent
  - If-Then-Else $(0 \le x, e_1, e_2)$ considered equivalent to $e_1$ if in current set of Examples x has only non-negative values
  - Only one representative among equivalent subexpressions needs to be considered for building larger expressions
- Fast and robust for learning expressions with ~ 15 nodes

# Symbolic Learning

- Use a constraint solver for both the synthesis and verification steps

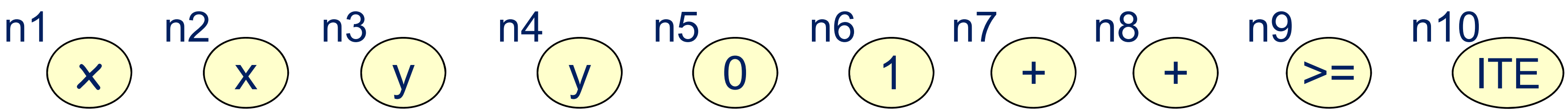
- Each production in the grammar is thought of as a component.
  Input and Output ports of every component are typed.




- A well-typed loop-free program comprising these components corresponds to an expression DAG from the grammar.

# Symbolic Learning

- Start with a library consisting of some number of occurrences of each component.




- Synthesis Constraints:
  - Shape is a DAG, Types are consistent
  - Spec φ[f/e] is satisfied on every concrete input values in Examples

- Use an SMT solver (Z3) to find a satisfying solution.

- If synthesis fails, try increasing the number of occurrences of components in the library in an outer loop

# Stochastic Learning

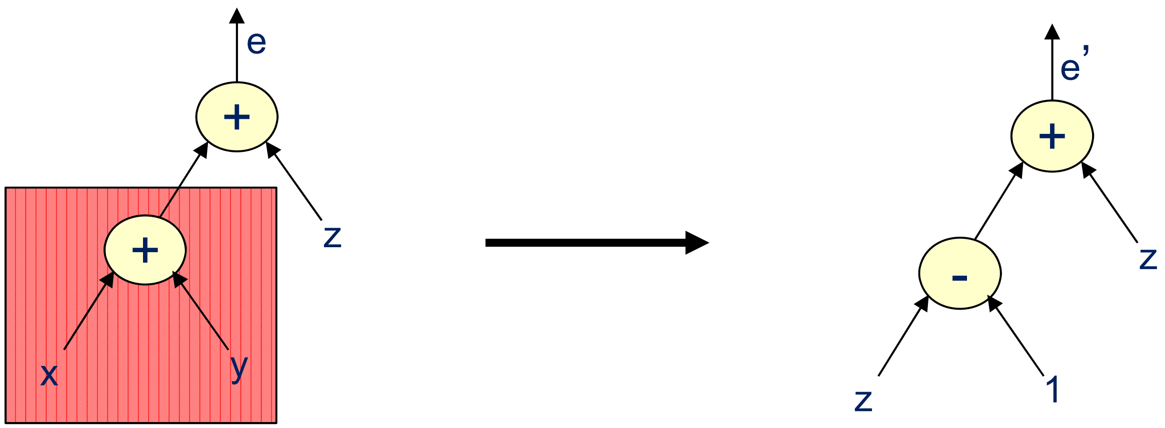
- Idea: Find desired expression e by probabilistic walk on graph where nodes are expressions and edges capture single-edits

- Metropolis-Hastings Algorithm: Given a probability distribution P over domain X, and an ergodic Markov chain over X, samples from X

- Fix expression size n.
  - X is the set of expressions $E_n$ of size n.
  - $P(e) \propto Score(e)$ ("Extent to which e meets the spec $\varphi$")

# Stochastic Learning

- Initial candidate expression e sampled uniformly from $E_n$
- If e works on all examples, return e
- Pick node v in parse tree of e uniformly at random. Replace subtree rooted at e with subtree of same size, sampled uniformly

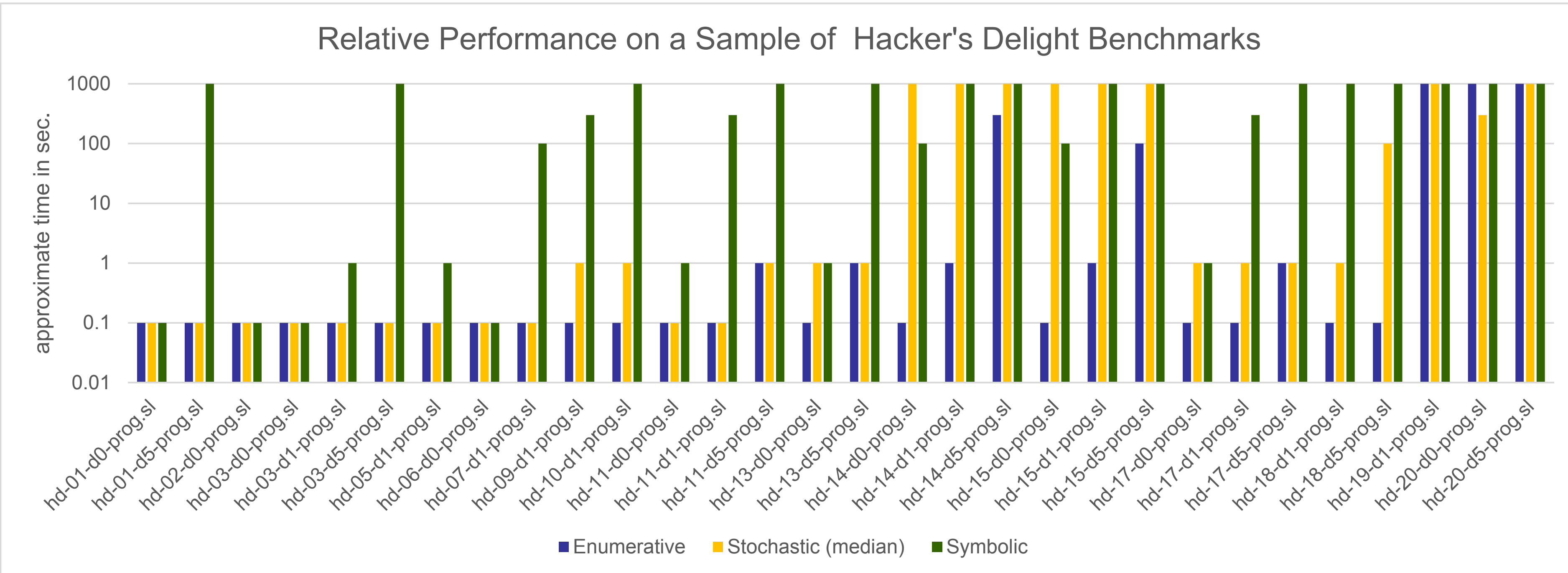



- With probability min{ 1, Score(e')/Score(e) }, replace e with e'
- Outer loop responsible for updating expression size n

# Benchmarks and Implementation

- Prototype implementation of Enumerative/Symbolic/Stochastic CEGIS
- Benchmarks:
  - Bit-manipulation programs from Hacker's delight
  - Integer arithmetic: Find max, search in sorted array
  - Challenge problems such as computing Morton's number
- Multiple variants of each benchmark by varying grammar
- Results are not conclusive as implementations are unoptimized, but offers first opportunity to compare solution strategies

# Evaluation: Hacker’s Delight Benchmarks



# Evaluation Summary

- Enumerative CEGIS has best performance, and solves many benchmarks within seconds

  Potential problem: Synthesis of complex constants

- Symbolic CEGIS is unable to find answers on most benchmarks

  Caveat: Sketch succeeds on many of these

- Choice of grammar has impact on synthesis time

  When E is set of all possible expressions, solvers struggle

- None of the solvers succeed on some benchmarks

  Morton constants, Search in integer arrays of size > 4

- Bottomline: Improving solvers is a great opportunity for research !