Adaptive Concretization for Parallel Program Synthesis

Jinseong Jeon[1], Xiaokang Qiu[2],
Armando Solar-Lezama[2], and Jeffrey S. Foster[1]

1. University of Maryland, College Park
2. MIT CSAIL

Syntax-guided Synthesis

```
bit[32] spec(bit[32] x) {
  return x - (x % 8);
}
```

program specification

```
bit[32] foo(bit[32] x)
        implements spec
{
  if (??) { // G
    return x & ??; // A
  } else {
    return x | ??; // B
} }
```

starts from structural hypothesis (a.k.a. *template*)

unknown 32-bit integer

Explicit Search

- Stochastic/systematic enumeration of candidate space

65 (=1 + 32*2)
unknown bits

```
bit[32] foo(bit[32] x)
        implements spec
{
  if (??) { // G
    return x & ??; // A
  } else {
    return x | ??; // B
} }
```

```
bit[32] foo(bit[32] x) ...
{
  if (true) { // G
    return x & 0x00000000; // A
  } else {
    return x | 0x00000000; // B
} }
```

```
bit[32] foo(bit[32] x) ...
{
  if (true) { // G
    return x & 0x00000001; // A
  } else {
    return x | 0x00000000; // B
} }
```

. . .

```
bit[32] foo(bit[32] x) ...
{
  if (false) { // G
    return x & 0xffffffff; // A
  } else {
    return x | 0xffffffff; // B
} }
```

Symbolic Search

- Constraint solving via SAT/SMT solver

```
bit[32] spec(bit[32] x) {
  return x - (x % 8);
}
```

Sketch solves in 50ms

```
bit[32] foo(bit[32] x)
        implements spec
{
  if (??) { // G
    return x & ??; // A
  } else {
    return x | ??; // B
} }
```

```
eq(spec(x),
  bvSub(x,
    bvMod(x, 8)))

eq(foo(x), spec(x))
eq(foo(x),
  ite(G,
    bvAND(x, A),
    bvOR(x, B)))
```

Adaptive Concretization

Explicit search | Adaptive Concretization | Symbolic search

```
bit[32] foo(bit[32] x) ...
{
  if (true) { // G
    return x & 0x00000000; // A
  } else {
    return x | 0x00000000; // B
} }
```

```
bit[32] foo(bit[32] x) ...
{
  if (true) { // G
    return x & 0x00000001; // A
  } else {
```

. . .

?

```
eq(spec(x),
  bvSub(x,
    bvMod(x, 8)))

eq(foo(x), spec(x))
eq(foo(x),
  ite(G,
    bvAND(x, A),
    bvOR(x, B)))
```

Symbolic Constraints

```
bit[32] spec(bit[32] x) {
  return x - (x % 8);
}
```

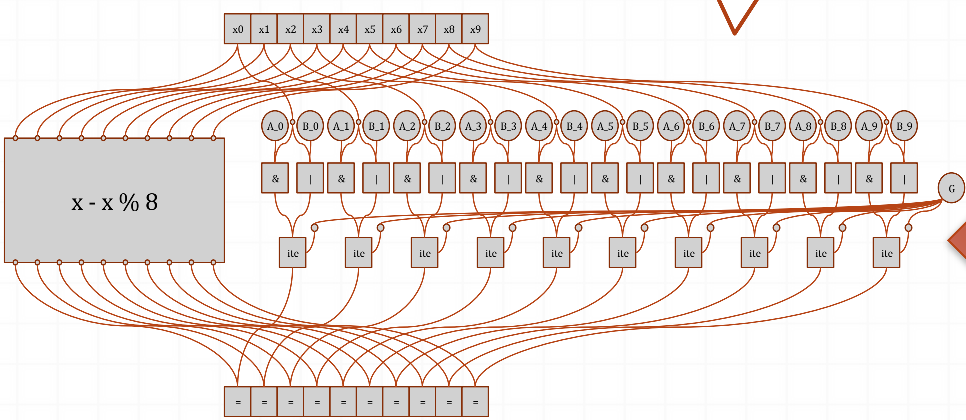
```
bit[32] foo(bit[32] x)
        implements spec
{
  if (??) { // G
    return x & ??; // A
  } else {
    return x | ??; // B
} }
```

```
eq(spec(x),
  bvSub(x,
    bvMod(x, 8)))

eq(foo(x), spec(x))
eq(foo(x),
  ite(G,
    bvAND(x, A),
    bvOR(x, B)))
```

Low-level SAT Formula

#node 488
50ms to solution

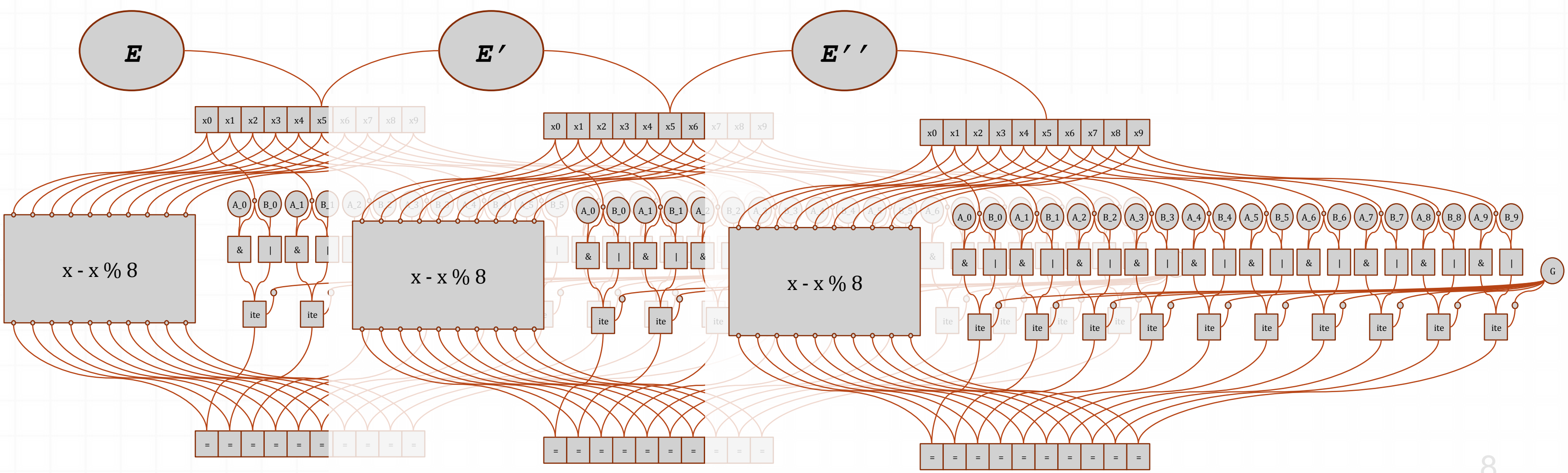



```
eq(spec(x),
  bvSub(x,
    bvMod(x, 8)))

eq(foo(x), spec(x))
eq(foo(x),
  ite(G,
    bvAND(x, A),
    bvOR(x, B)))
```

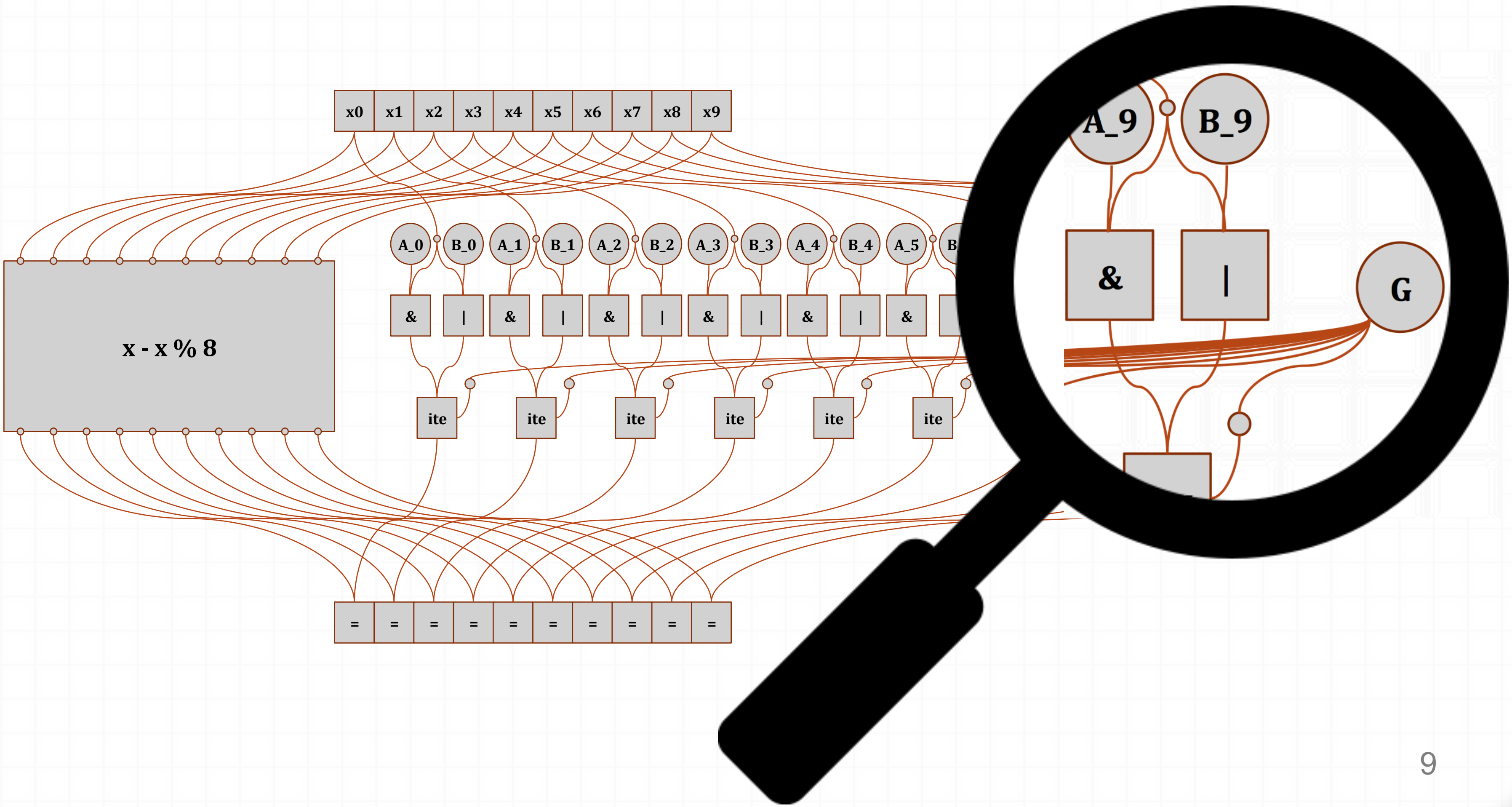
CEGIS

- Synthesis: $\exists c. \forall x. Q(x, c)$
- Counter-Example Guided Inductive Synthesis
  - All x ⇒ x_i in E $\bigwedge_{x_i \in E} Q(x_i, c)$
  - Candidate solution for c is checked
  - Counter example is added to E; and repeat
- Reducing the formula Q is a big win



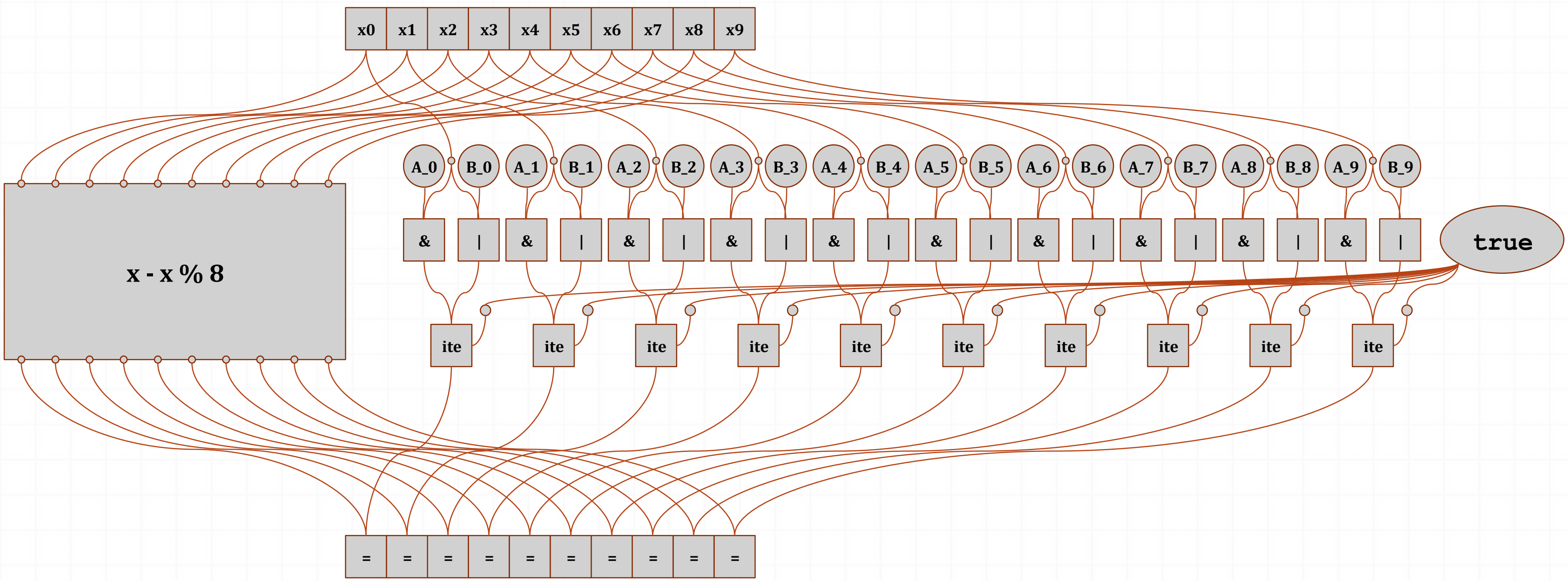
Highly Influential Unknowns

- Key observation: Unknowns are not all equally important



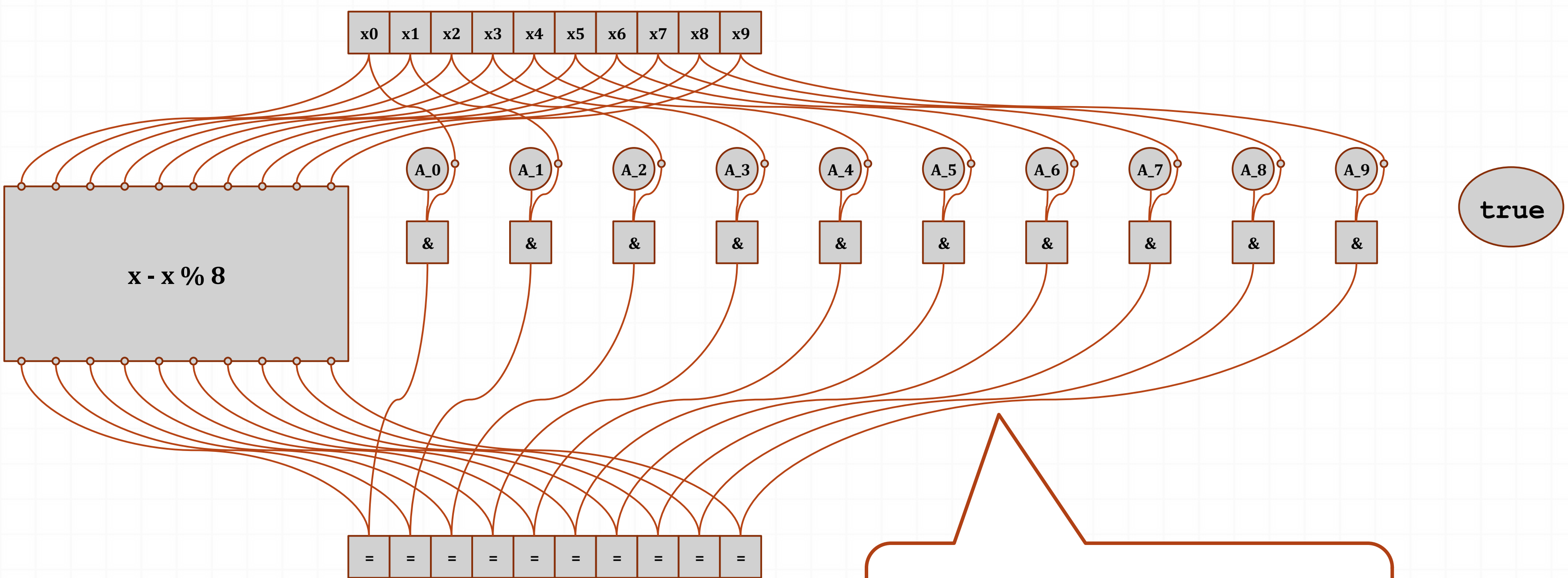
Partial Concretization

- Replacing an unknown with a concrete value



Partial Concretization

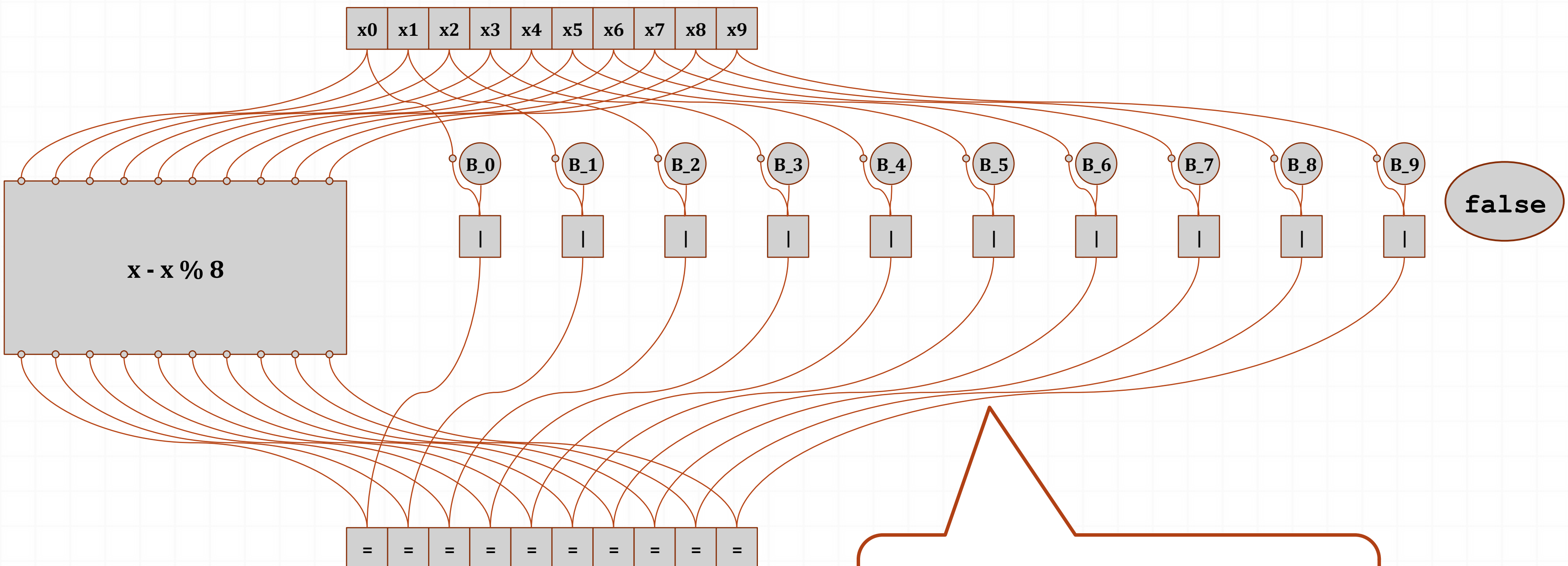
- Then simplifying the formula



#node 488 ⇒ 391
50 ⇒ 30ms to solution

Partial Concretization

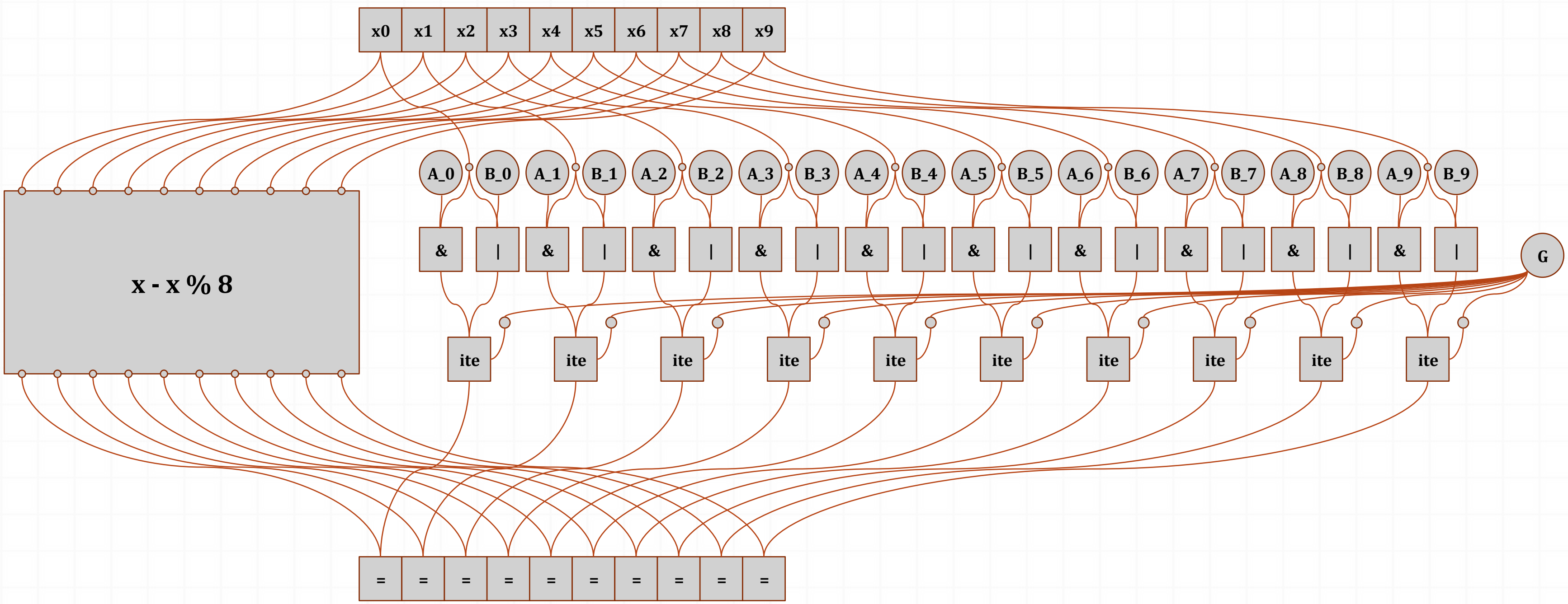
- Beneficial even with a wrong concrete value
  - Running two trials (incorrect one and then correct one) is faster than pure symbolic search



#node 488 ⇒ 391
50 ⇒ 2ms to UNSAT

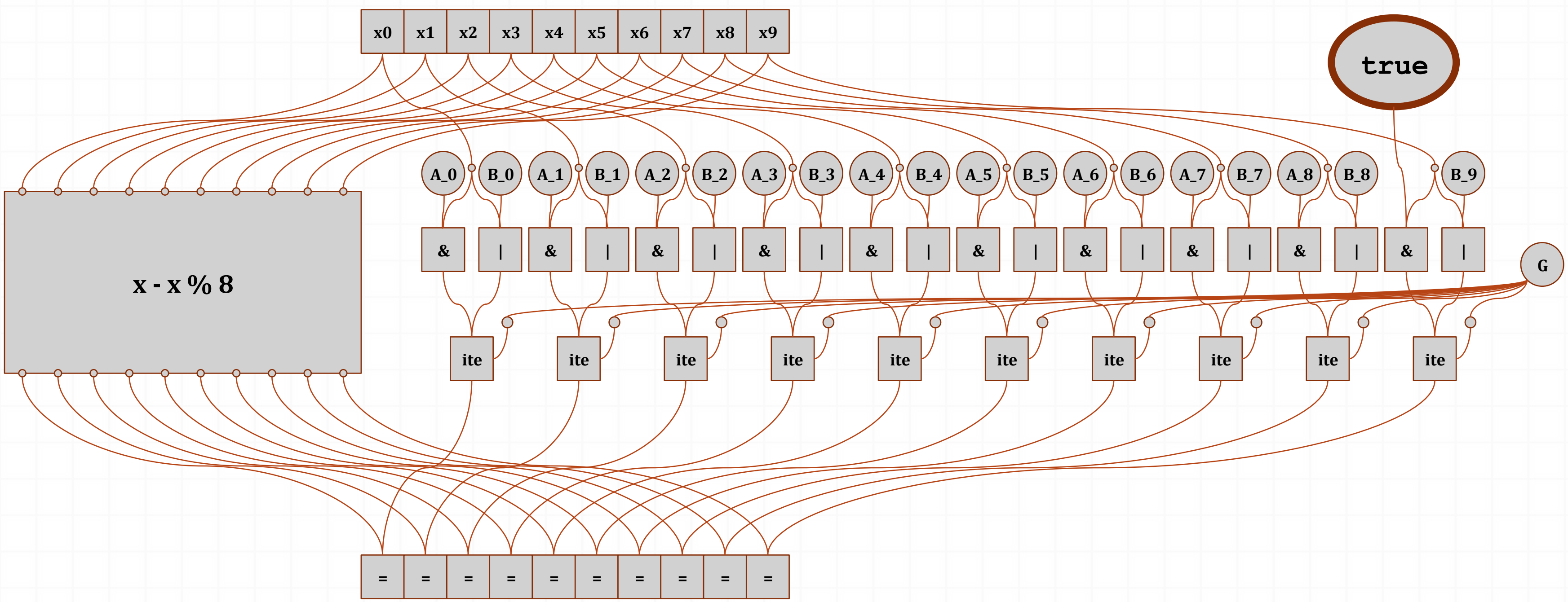
Influence of Unknowns

- Key observation: Unknowns are not all equally important



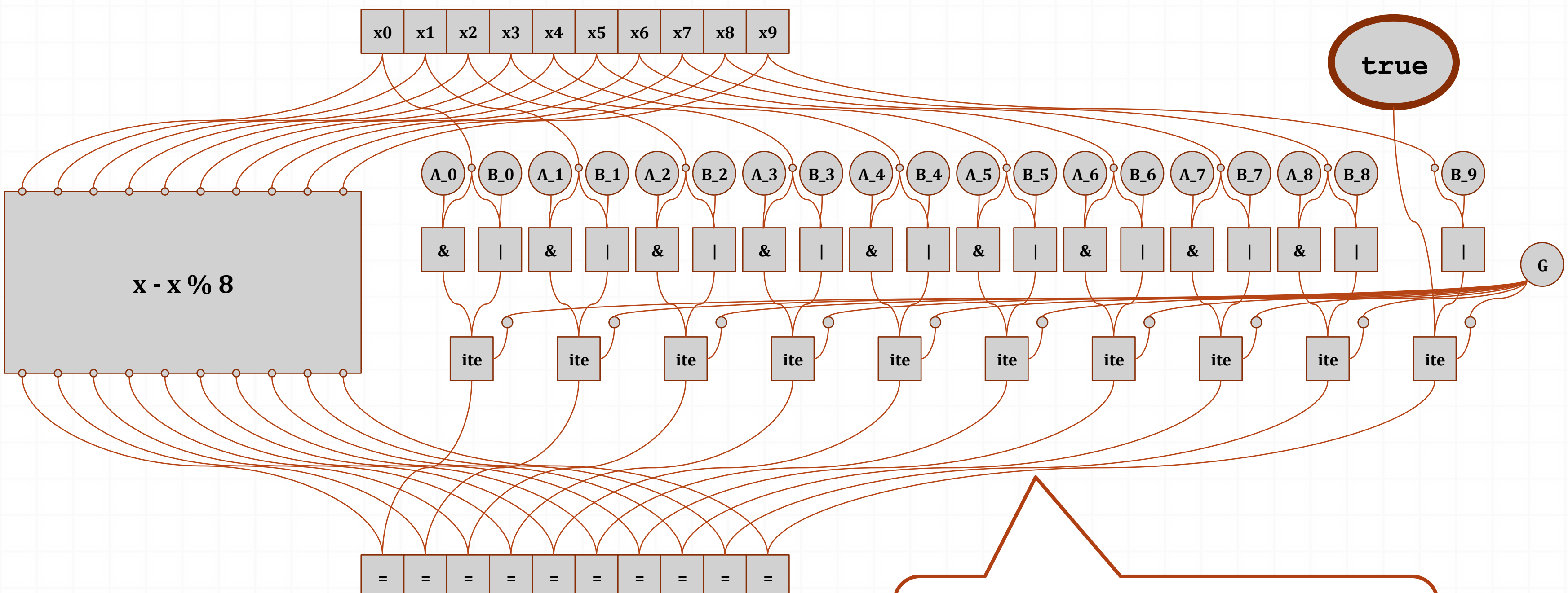
Unknowns for Computation

- Arithmetic unknowns are best left to the solver



Unknowns for Computation

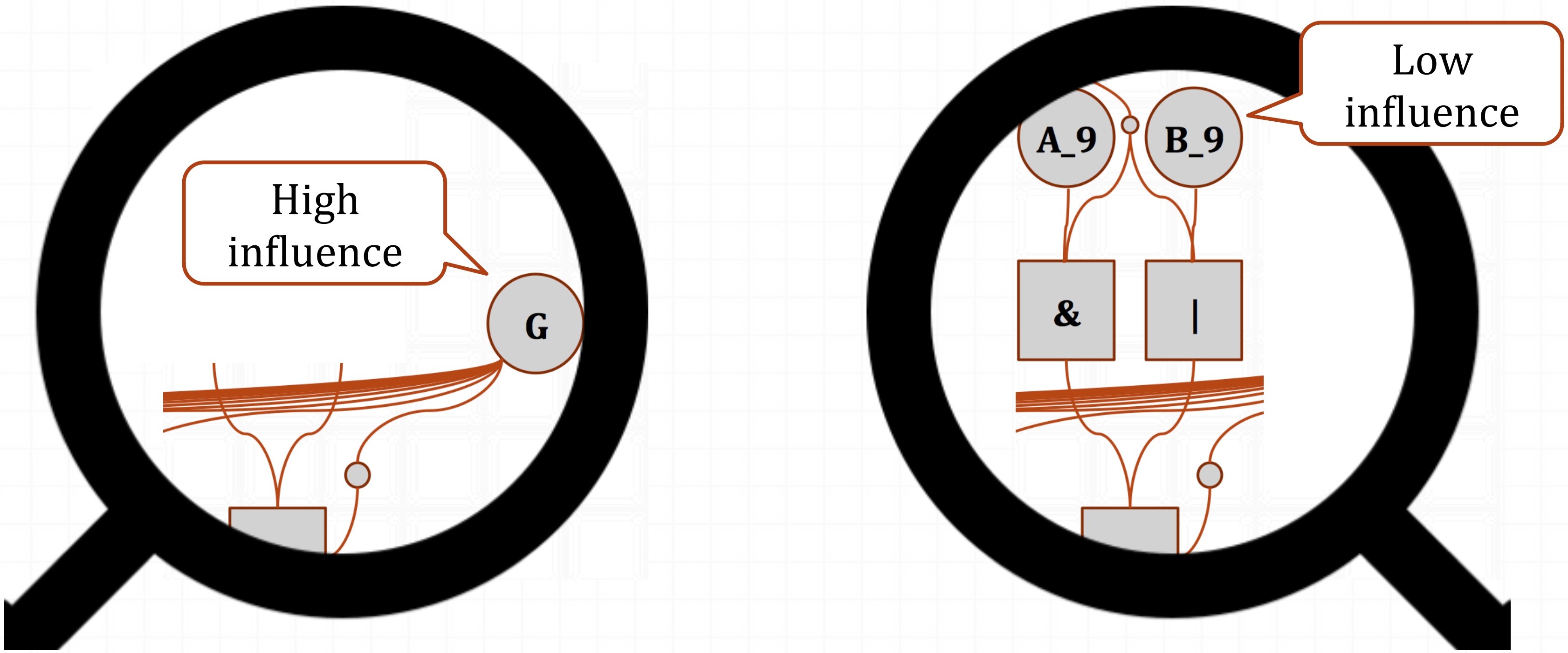
- Arithmetic unknowns are best left to the solver



#node 488 ⇒ 486
50ms to solution/UNSAT

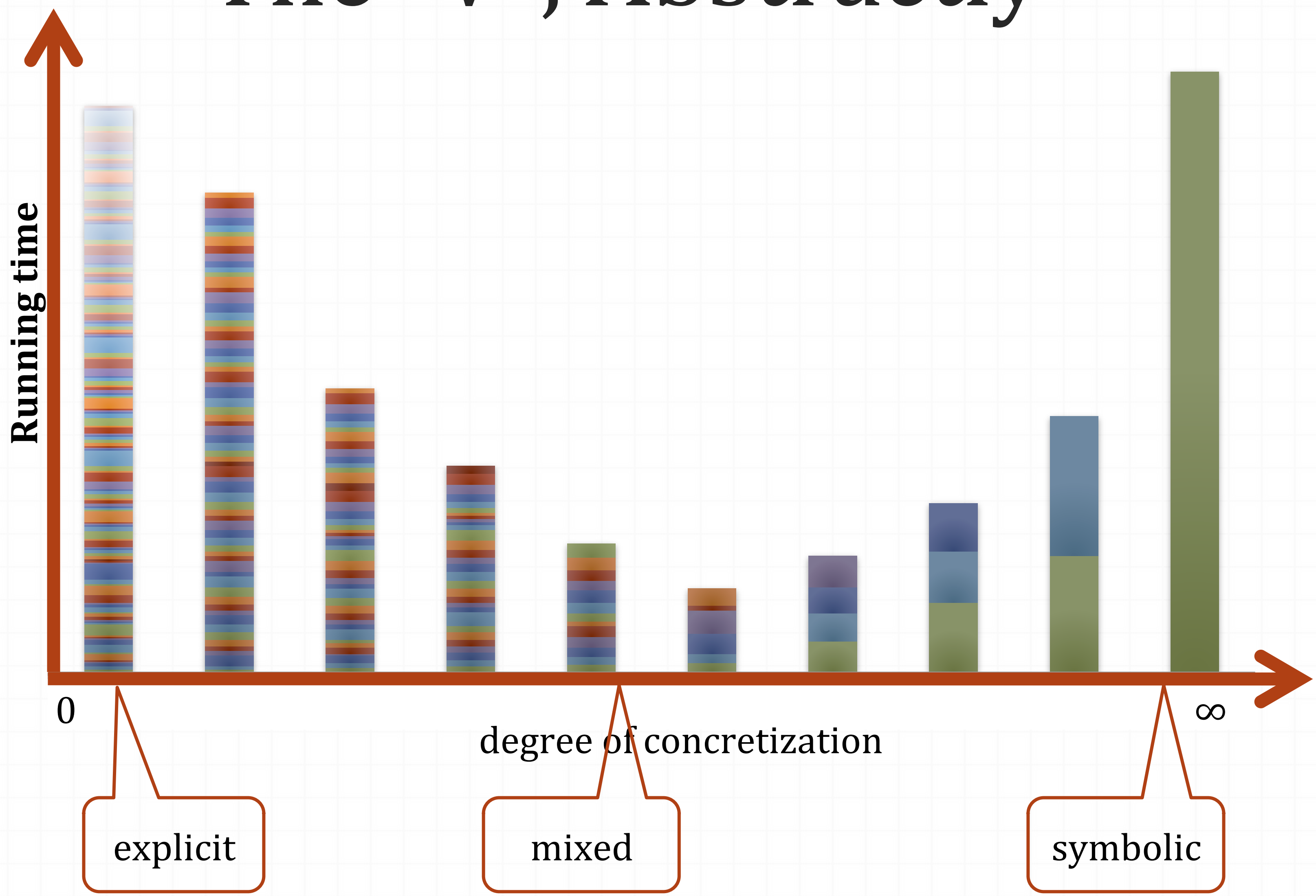
Influence Estimation

- Key observation: Unknowns are not all equally important

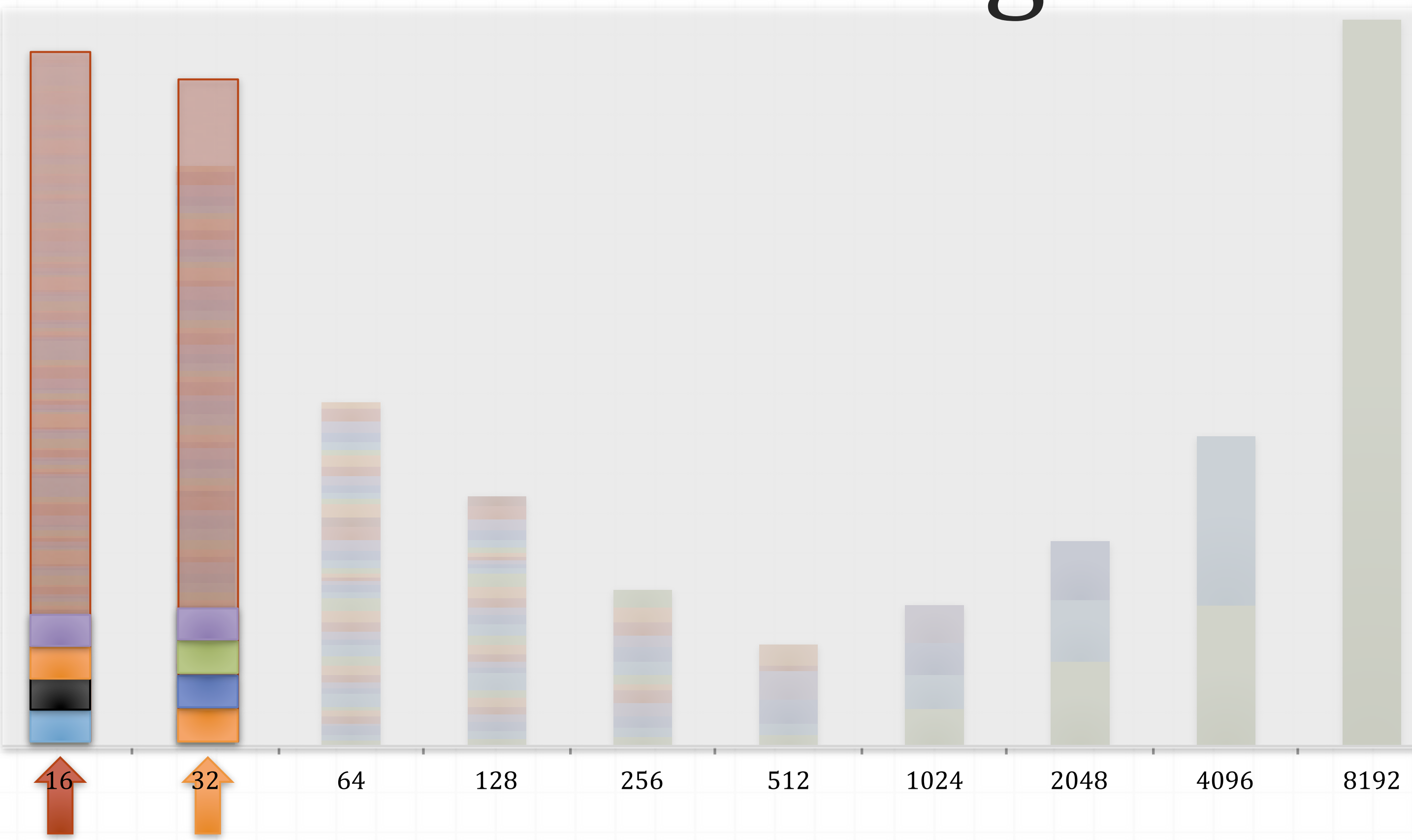


- But, this influence computation is an *estimate*.
- We opt to *randomize*!

The “V”, Abstractly

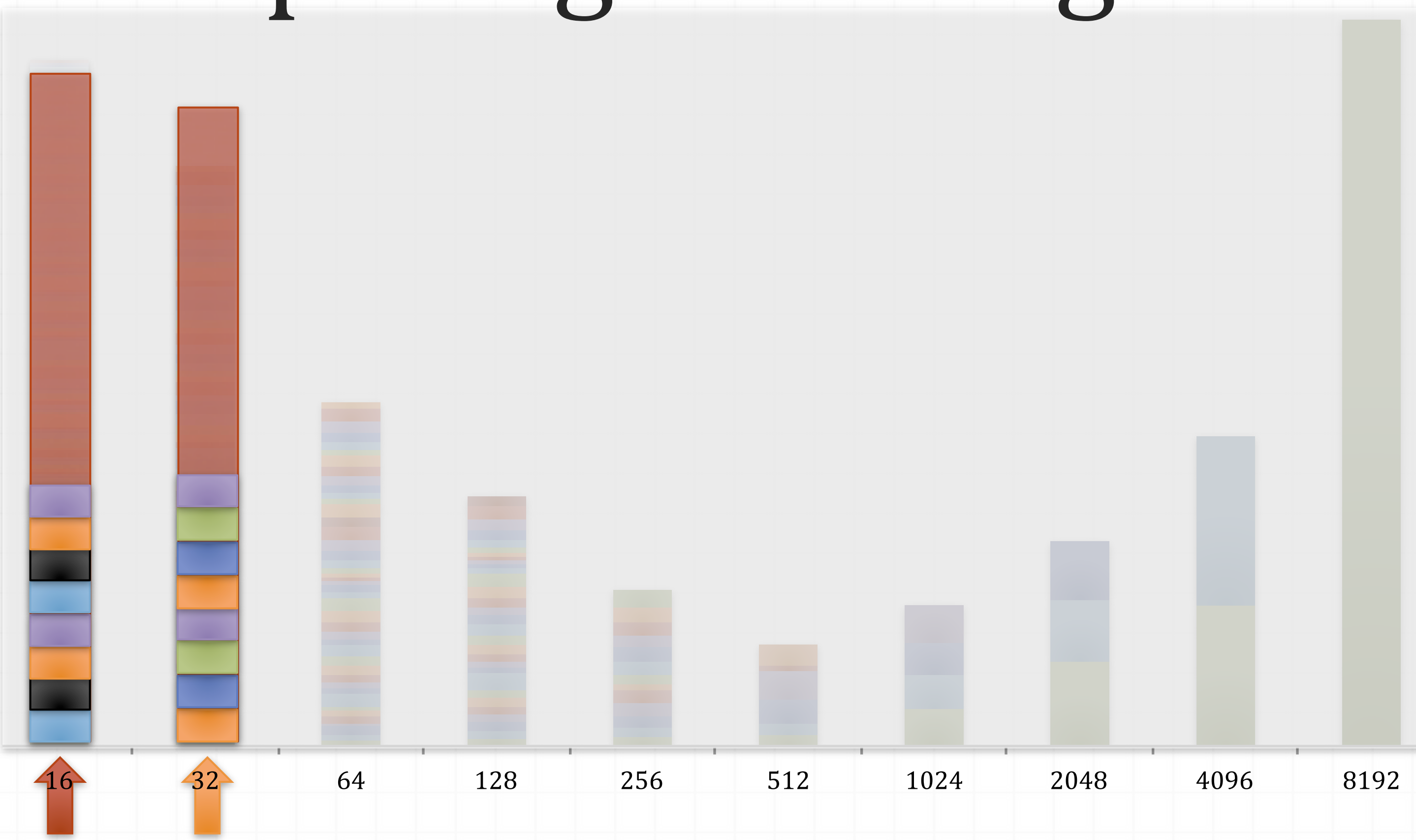


Estimated Running Time



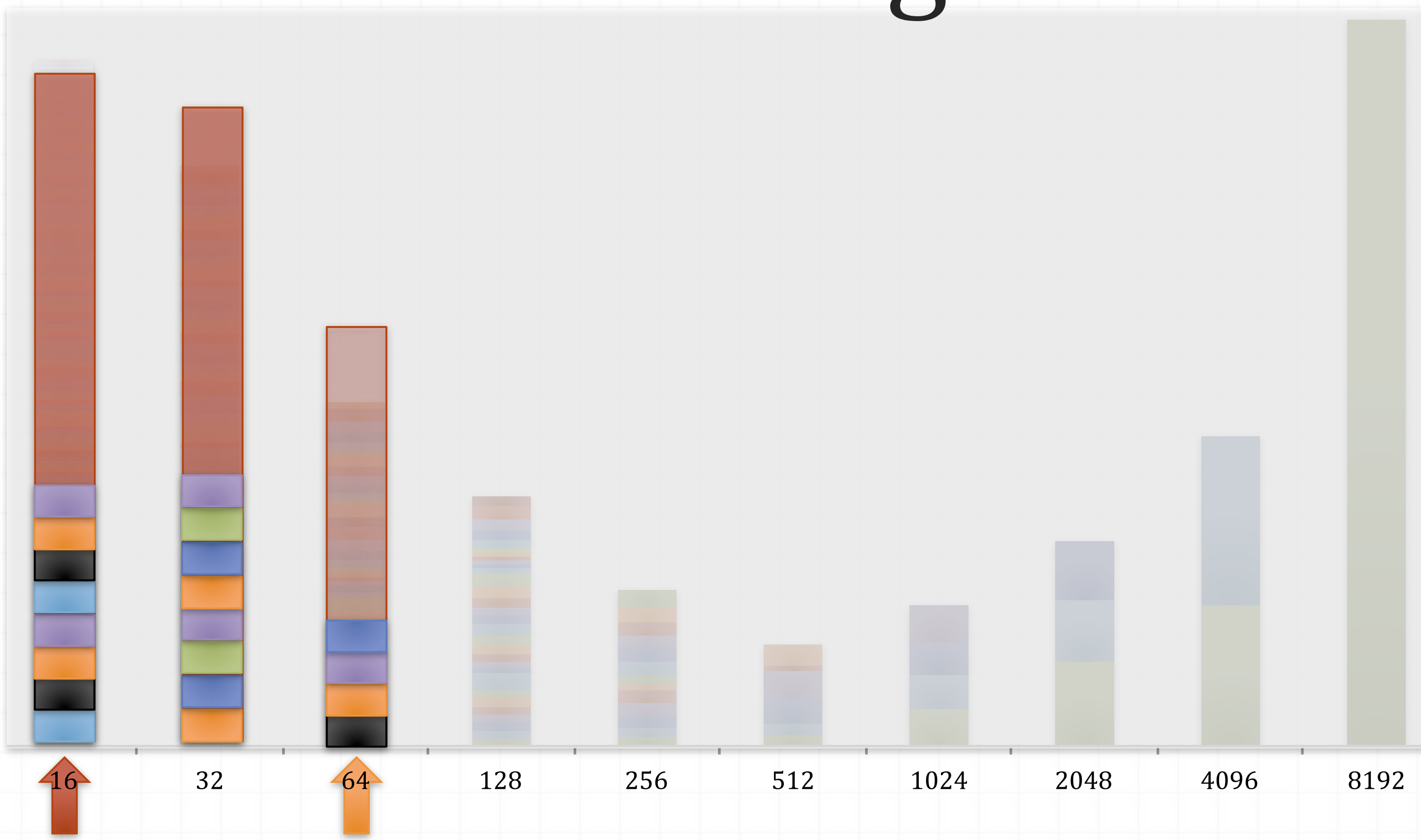
- Starts from low degrees (to avoid long-running high degree)
- Runs trials (in parallel) and estimates running time

Comparing Two Degrees



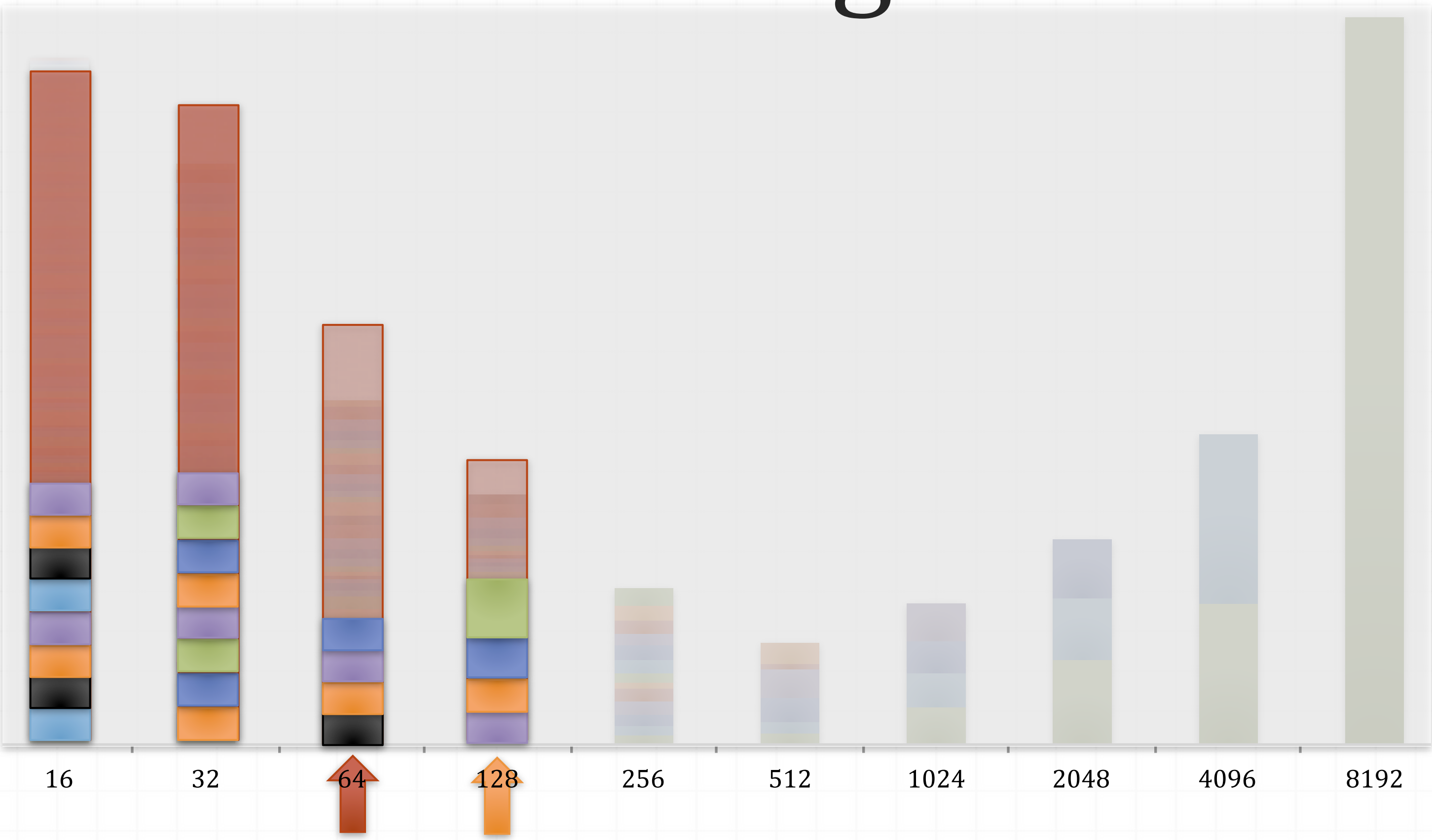
- Wilcoxon Signed-Rank Test
  - determines if two data sets are from distinct populations

Widening



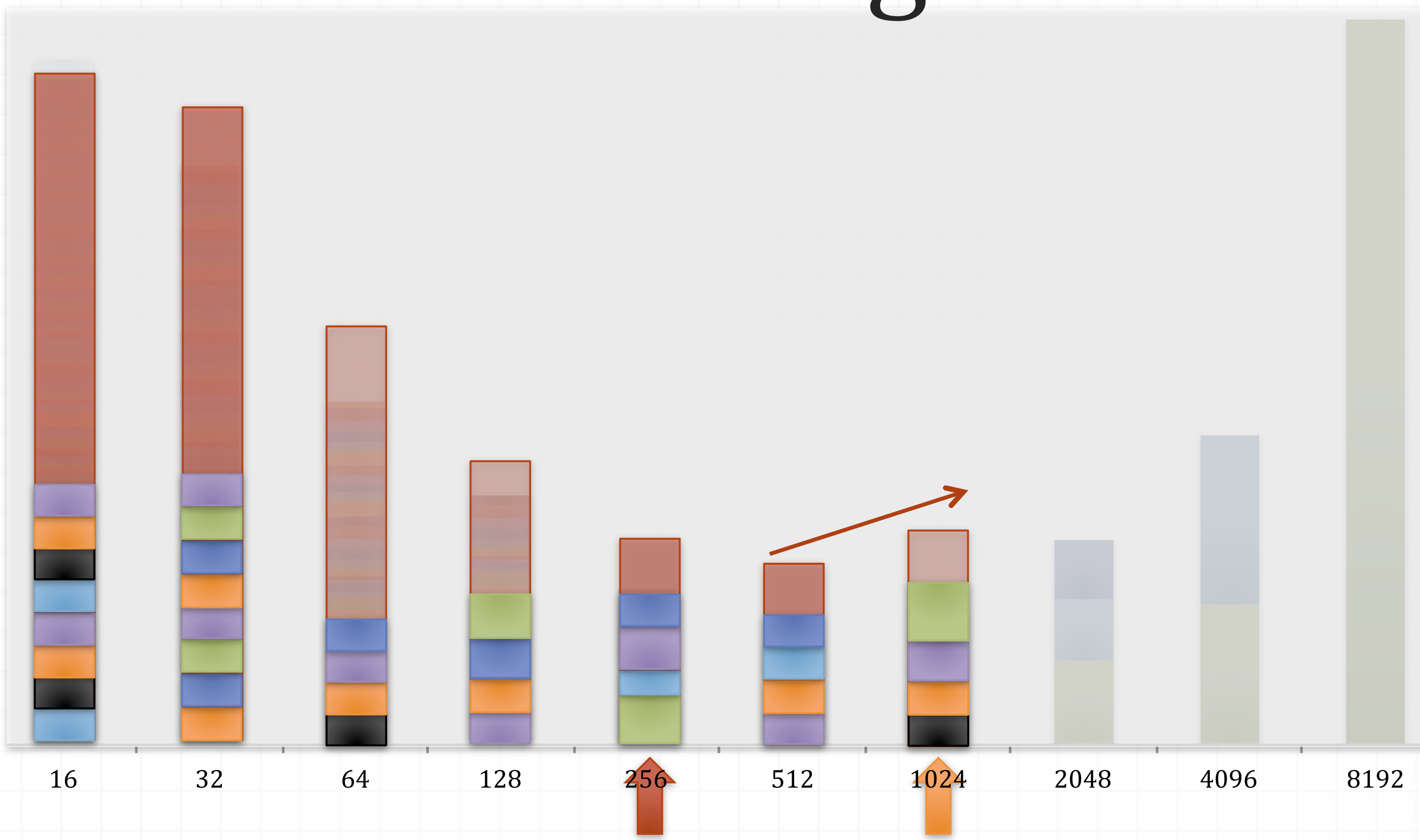
- Widen the range
  - if not distinguishable

Shifting



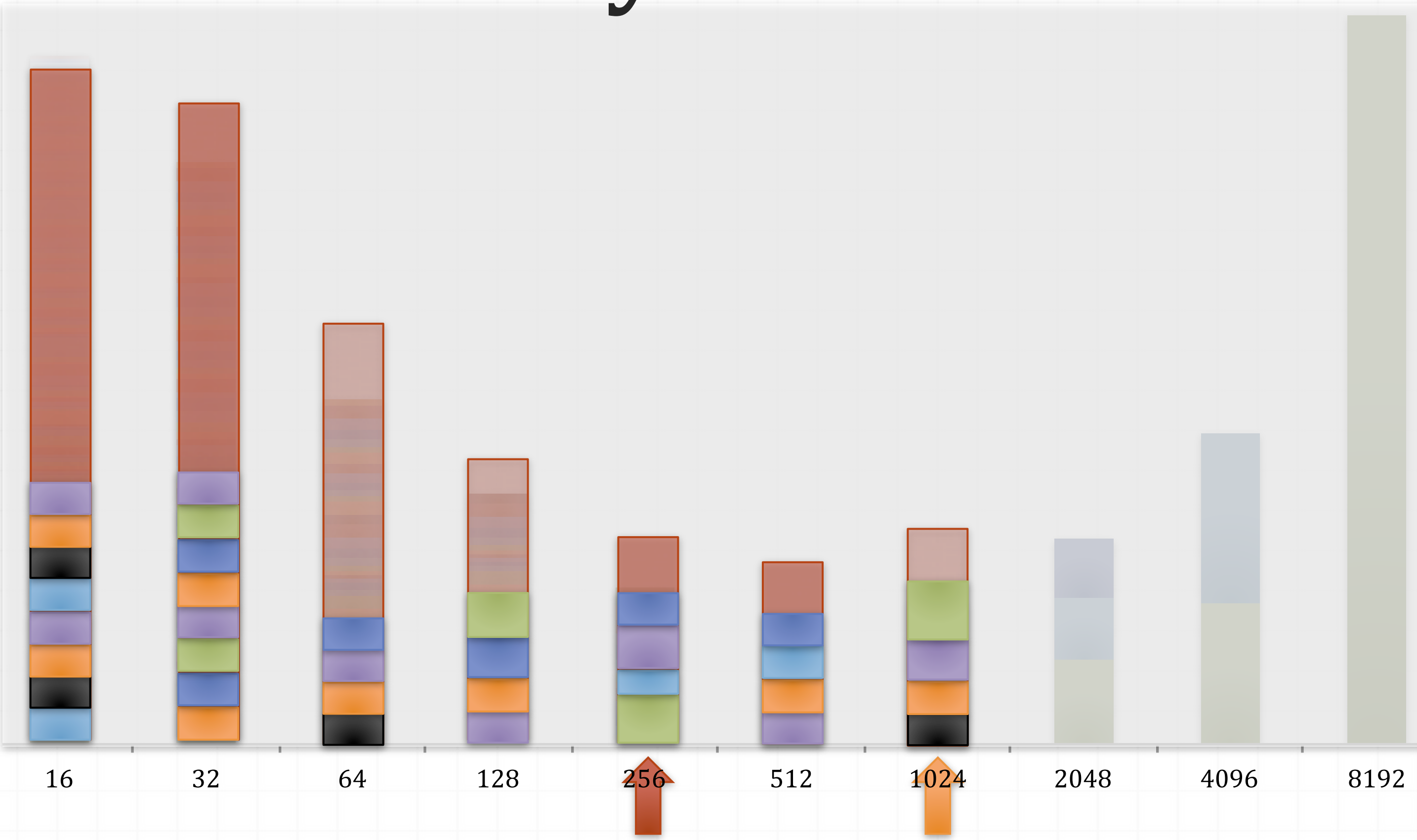
- Shift to the next range
  - if distinguishable and the high pivot is faster (i.e., down-hill)

Climbing



- Keep climbing until the up-hill appears
  - optimum likely lies between that range

Binary Search



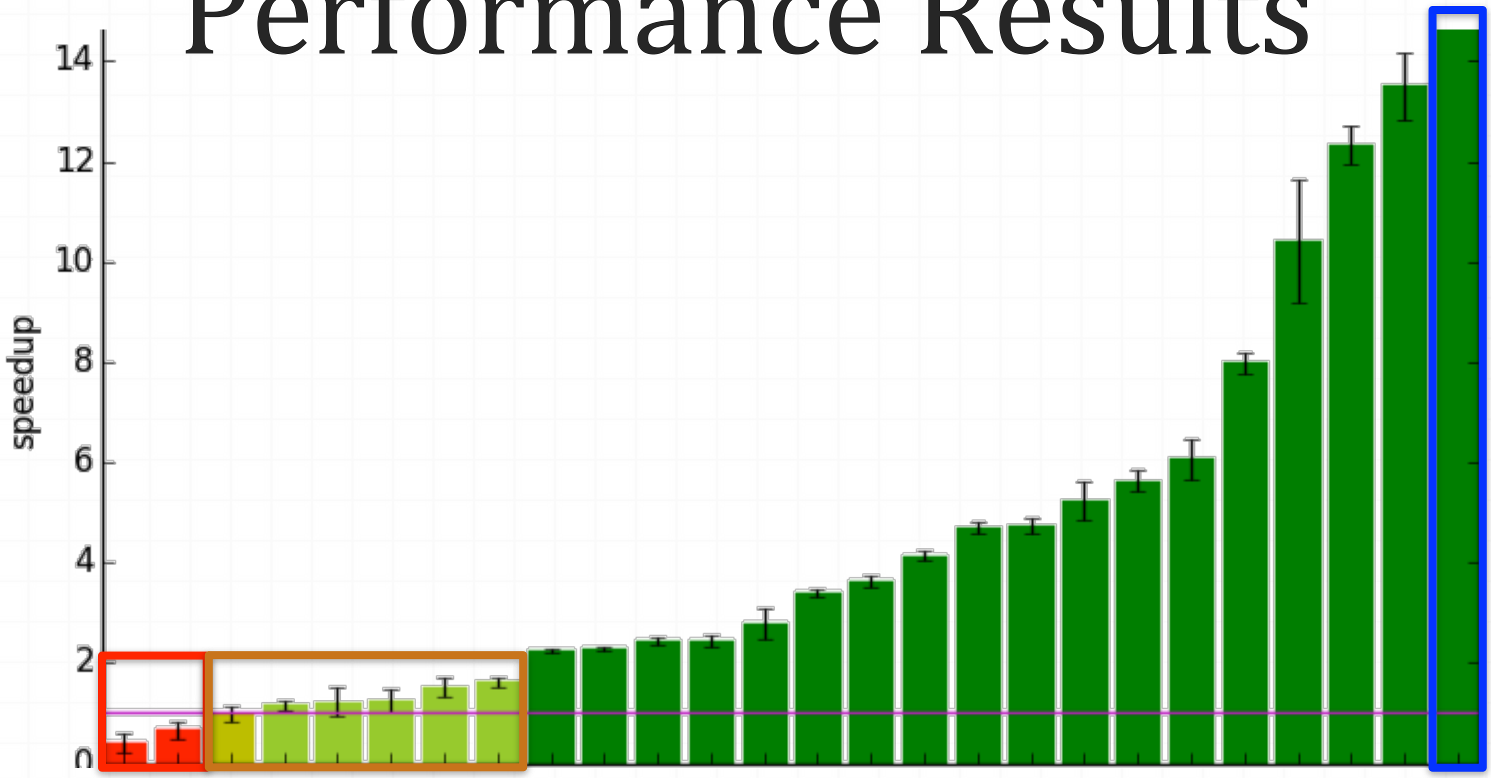
- Binary search between the rough range

Experiment

- 26 sketches from 5 domains
  - Pasket – framework model synthesis
    - The motivation for this work
    - Several Pasket examples could not run under plain Sketch
  - Data structure manipulation
  - Invariants for stencils - Scientific computation
  - SyGuS 2014
  - Sketch performance benchmarks

- Did 13 runs on server with forty 2.4GHz CPUs, 99GB RAM, Ubuntu 14.04.1 LTS
  - 2-hour timeout, 32GB memory bound

Performance Results



- Running on 32 cores, compared to Sketch
- Adaptive Concretization (AC) better on 23 of 26
  - In one case, Sketch often aborts with out of memory
  - Many cases with speedups from 3x to 14x

Parallel Scalability Results

- Run on 1, 4, and 32 cores
  - AC faster on 1-core in 17 of 26 cases
  - AC generally speeds up with more cores
    - Best performance at 32 cores in 20 of 26 cases
  - Implementation does not fully utilize cores
    - Current source of overhead: re-loading input file every time

- Compared to Enumerative Solver (SyGuS 2014 winner)
  - Faster on 6 of 9 benchmarks
  - Competitive on others

Conclusion

- Adaptive Concretization
  - Concretized high influence unknowns
  - Degree of concretization controls probability of concretizing
  - Parallel synthesis algorithm

- Key results
  - Degree of concretization varies with problem
  - Adaptive concretization much faster than Sketch
  - Reasonable parallel scalability

- http://plum-umd.github.io/adaptive-concretization/