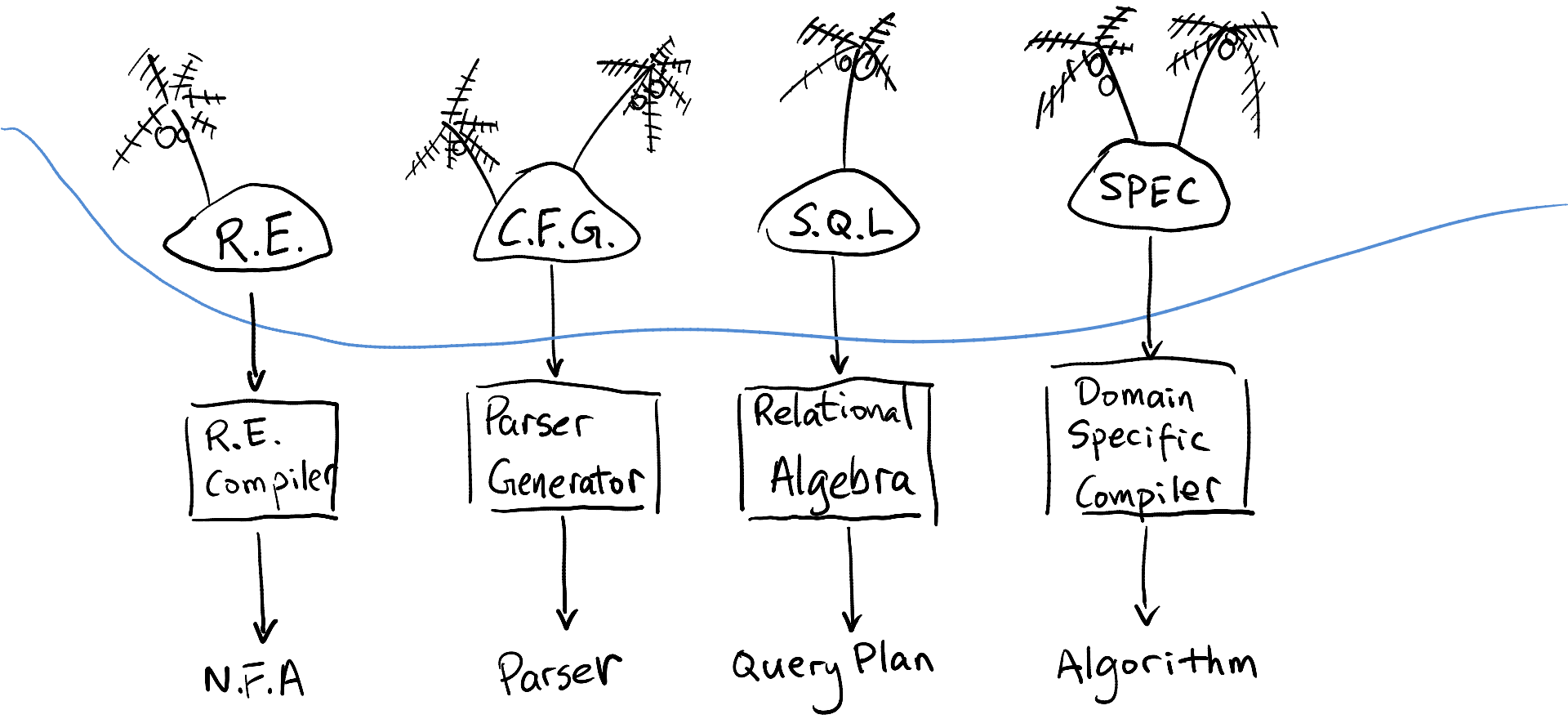
Synthesis Of First-Order Dynamic Programming Algorithms

Yewen (Evan) Pu
Rastislav Bodik
Saurabh Srivastava

University of California, Berkeley

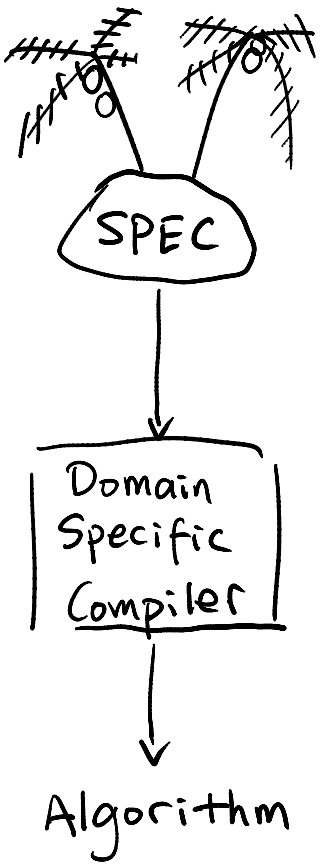
Do you feel lucky?



A new way to build an island?

Conventional Domain Specific Compiler:

- Require deep domain theories
- Takes a long time to implement



Constraint Based Synthesizer:

- Write a template for desired algorithm
- Constraint solver fills in the template

Make Templates not Theories

`Suppose we want to optimize x+x+x+x`

With domain-specific **rewrite rules:**

$x + x + x + x \longrightarrow 4 * x \longrightarrow 2^2 * x \longrightarrow x << 2$

With a template in SKETCH [Solar-Lezama]:

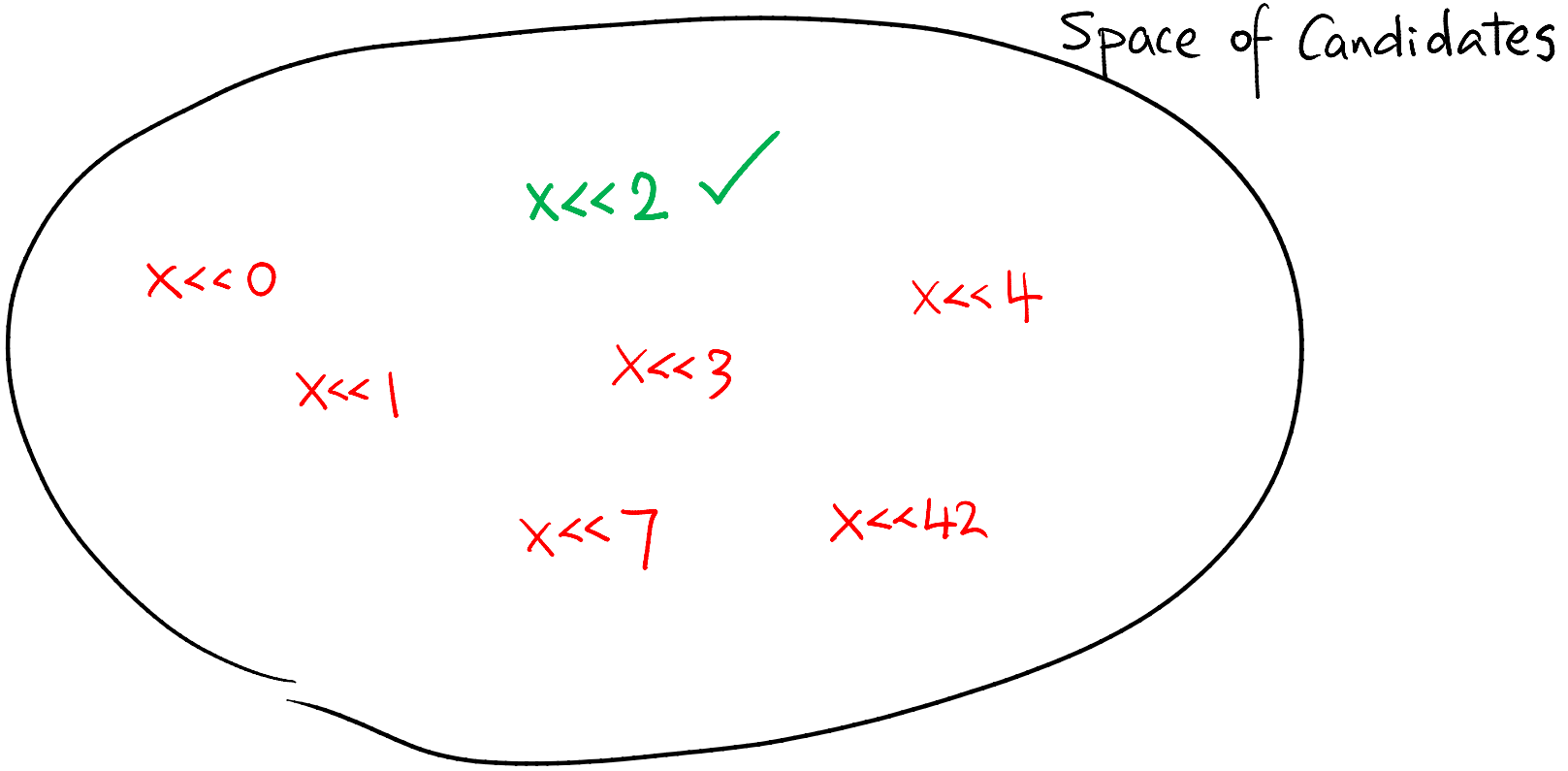
```
spec(x): return x+x+x+x

sketch(x): return x << ?? 2
```

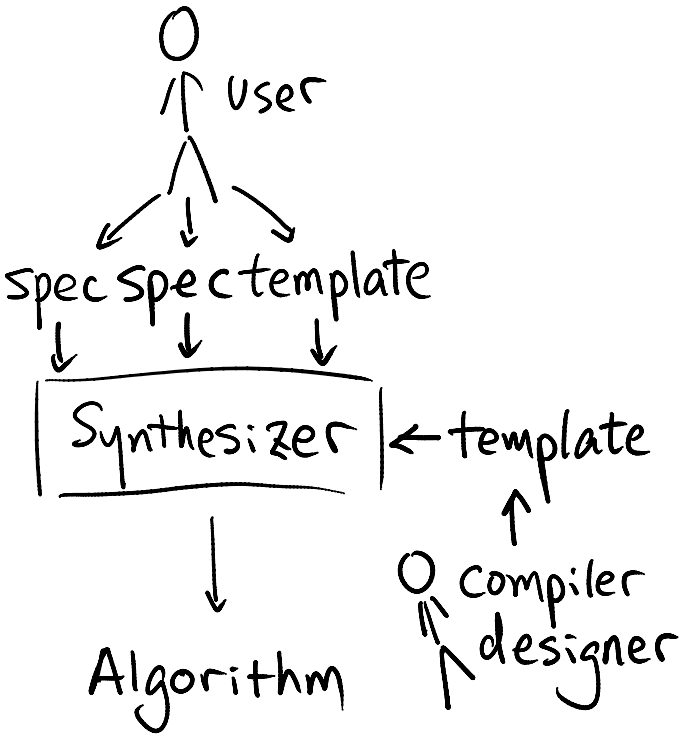
program equivalence found
using bounded model checking

A Search for a Correct Program

Synthesizer finds in a space of candidate programs a correct one (it matches the specification)



Research Question in This Talk



Can we write a synthesizer for an entire problem domain using a single template (sketch)?

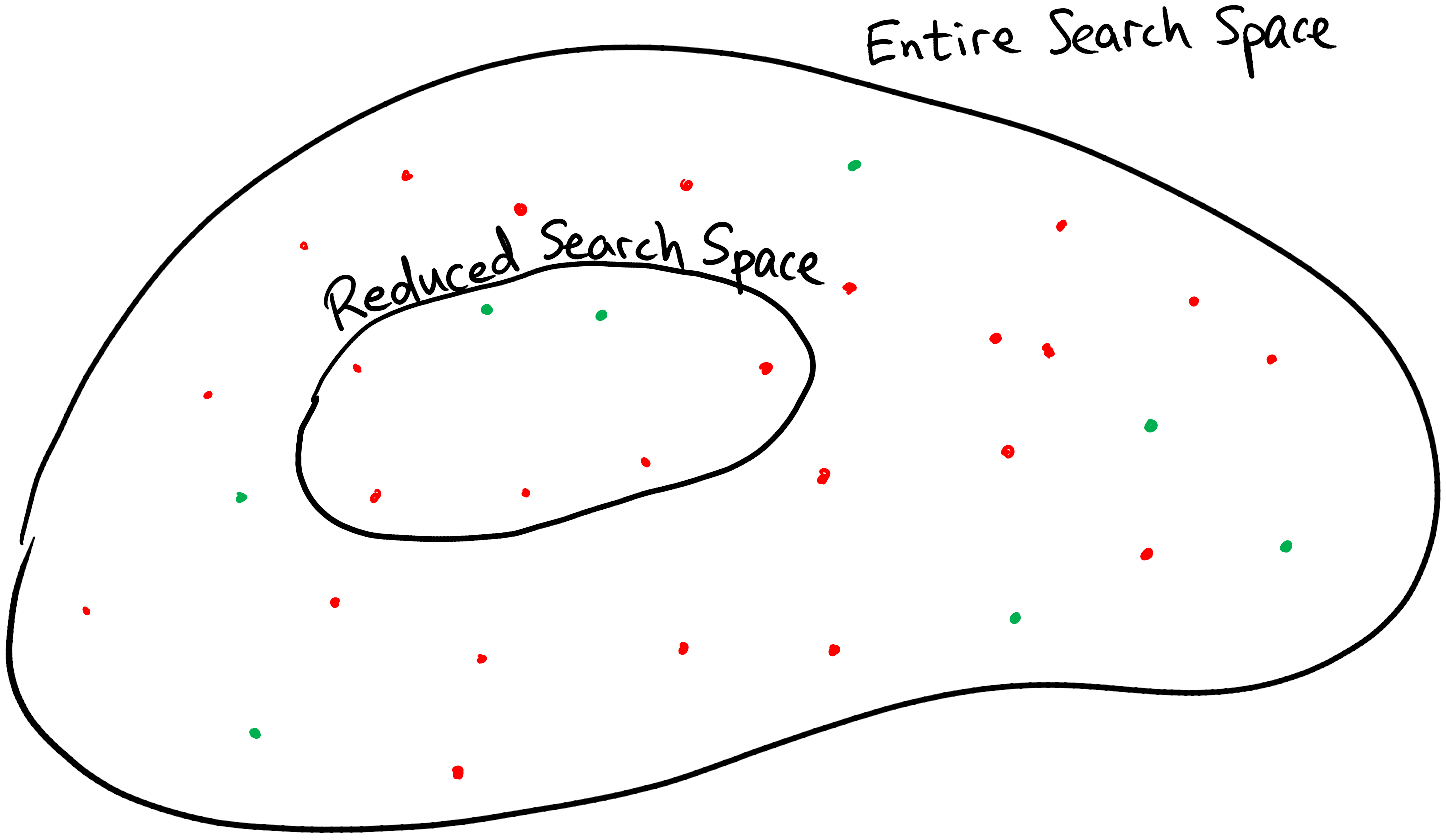
The Challenge

How do we write a template that covers all of our domain, yet the constraints it induces can be efficiently solved?

Our Approach

Define a general template that contains the entire domain

Optimize the search by reducing the search space



Dynamic Programming Algorithms

A well-defined domain

We have no “DSL compiler” for it (taught as an art)

Difficulties:

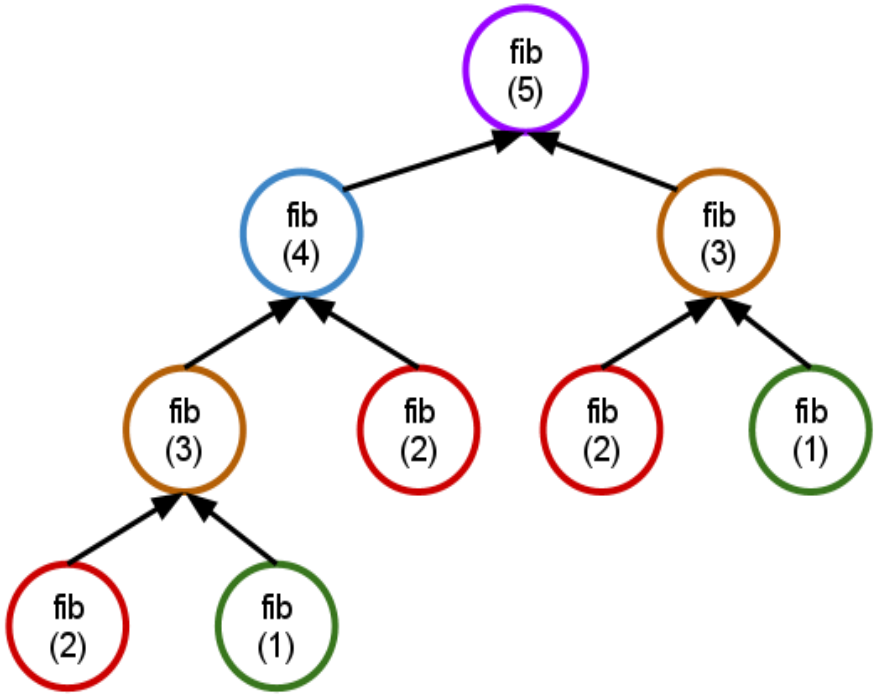
- inventing sub-problems
- inventing recurrences

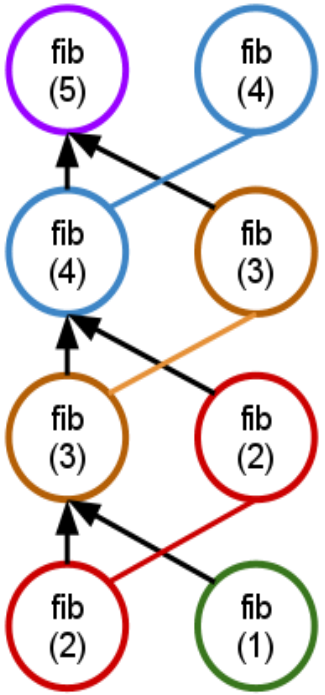
We focus on a first-order sub-class, FORDP, which captures many O(n) DP algorithms

An Easy Problem

Fibonacci Sequence:

fib(n) = fib(n-1) + fib(n-2)





A Harder Problem

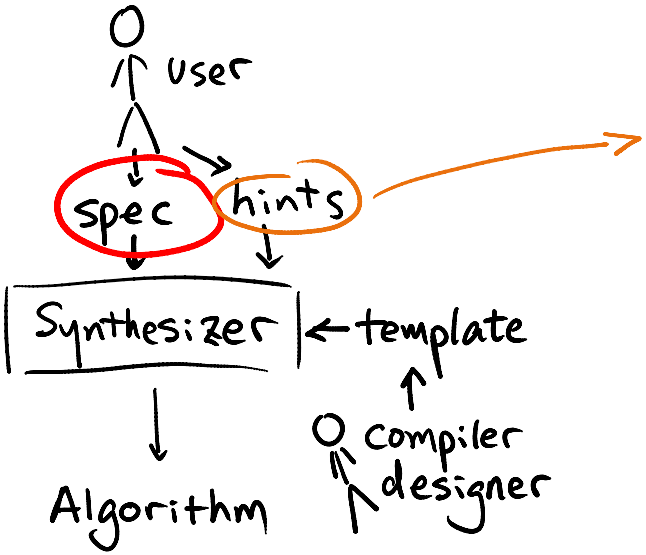
Maximal Independent Sum (MIS)

Input: Array of positive integers

Output: Maximum sum of a non-consecutive selections of its elements.

MIS([6,2,3,5,1,7,3]) = 18

What does the user do?



Hints for MIS:

operators:

zero,

identity,

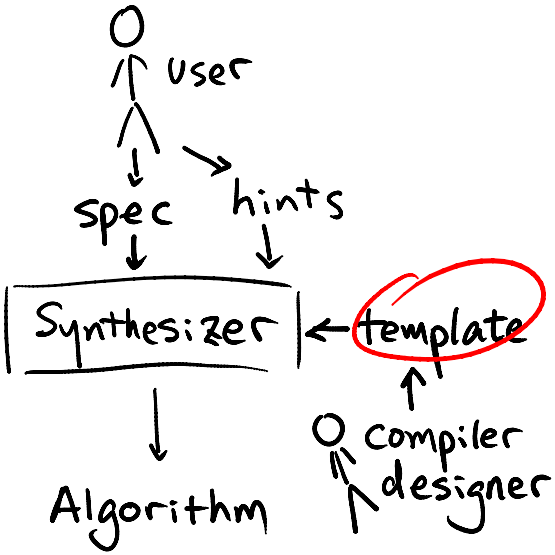
+,

max

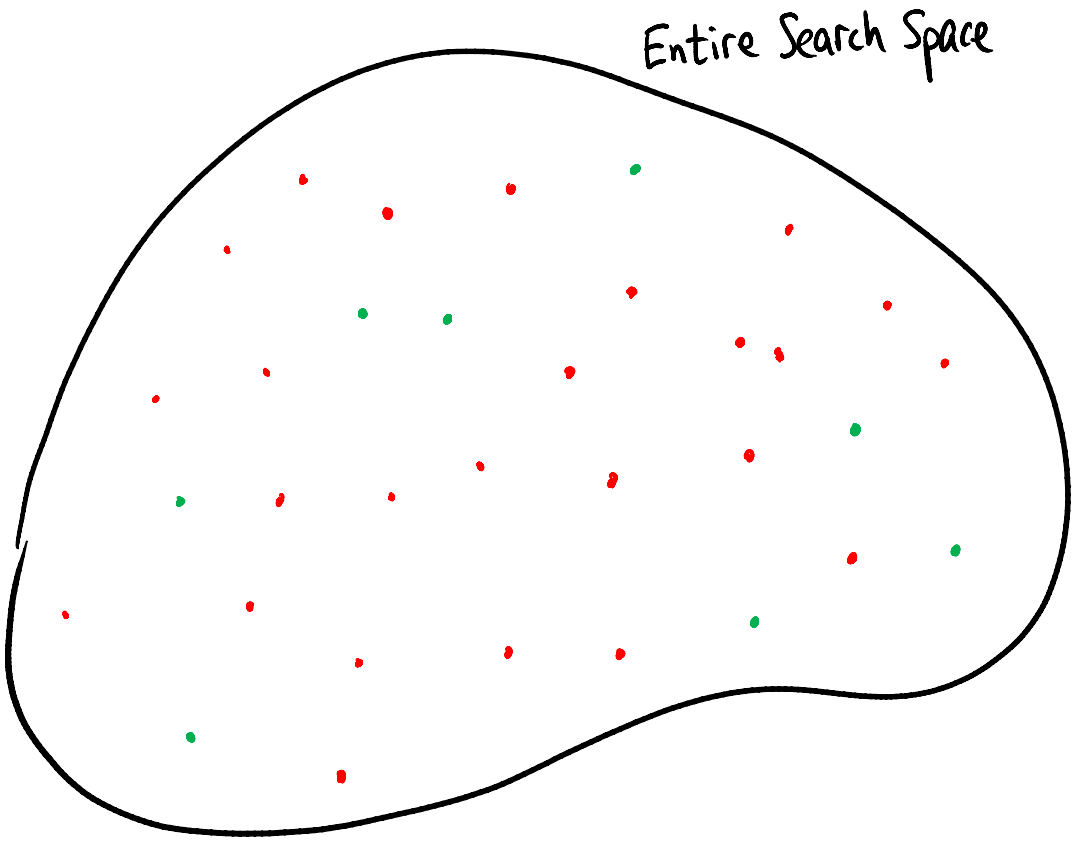
```
mis(A):
    best = 0
    forall selections:
        if non_consec(selection):
            best = max(best, value(A[selection]))
    return best
```

Does not reason about sub-problem

What does the template do?



- Define a general template that contains the entire domain



A General Template

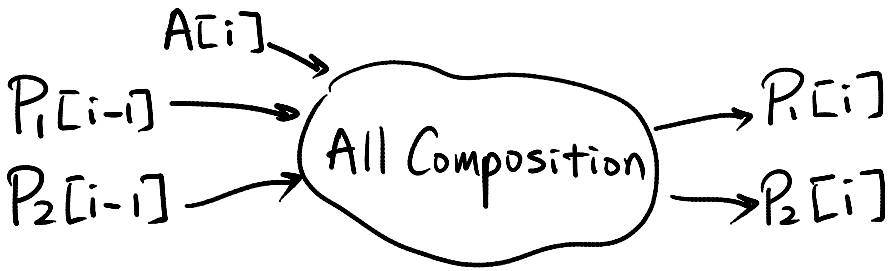
```
for_dpa(a):
   p1 = array()
   p2 = array()
   p1[0] = init1()
   p2[0] = init2()
   for i from 1 to n:
    p1[i] = update1(p1[i-1],p2[i-1],a[i])
    p2[i] = update2(p1[i-1],p2[i-1],a[i])
   return term(p1[n],p2[n])
```

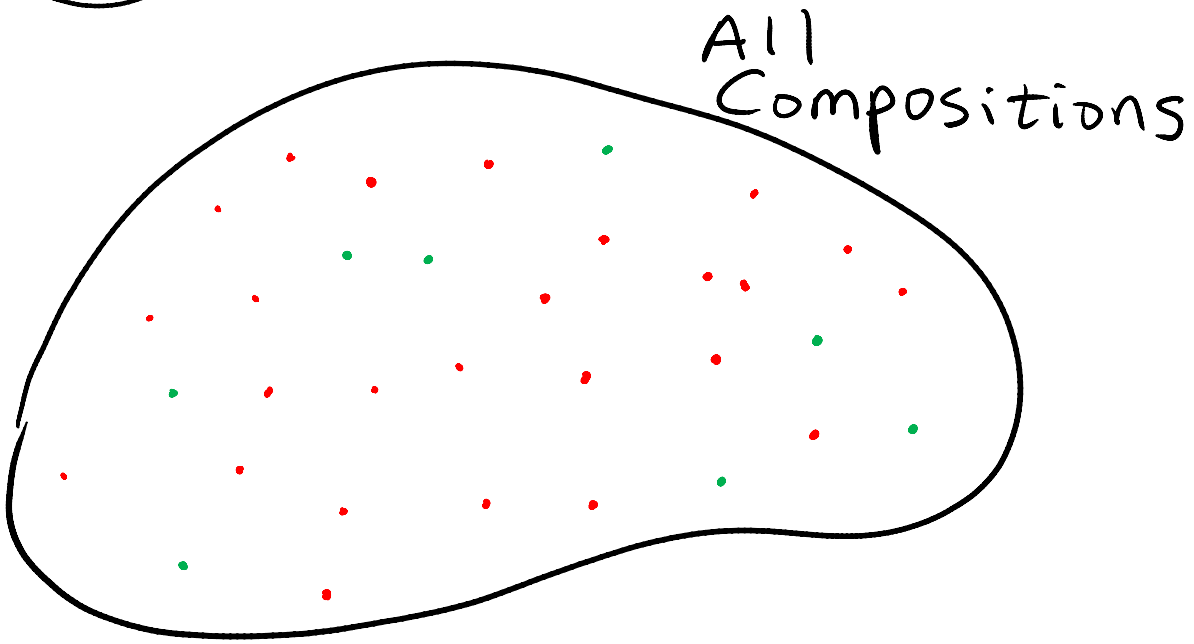
Defines the recurrence

Covers every FORDP algorithm

General Template for update

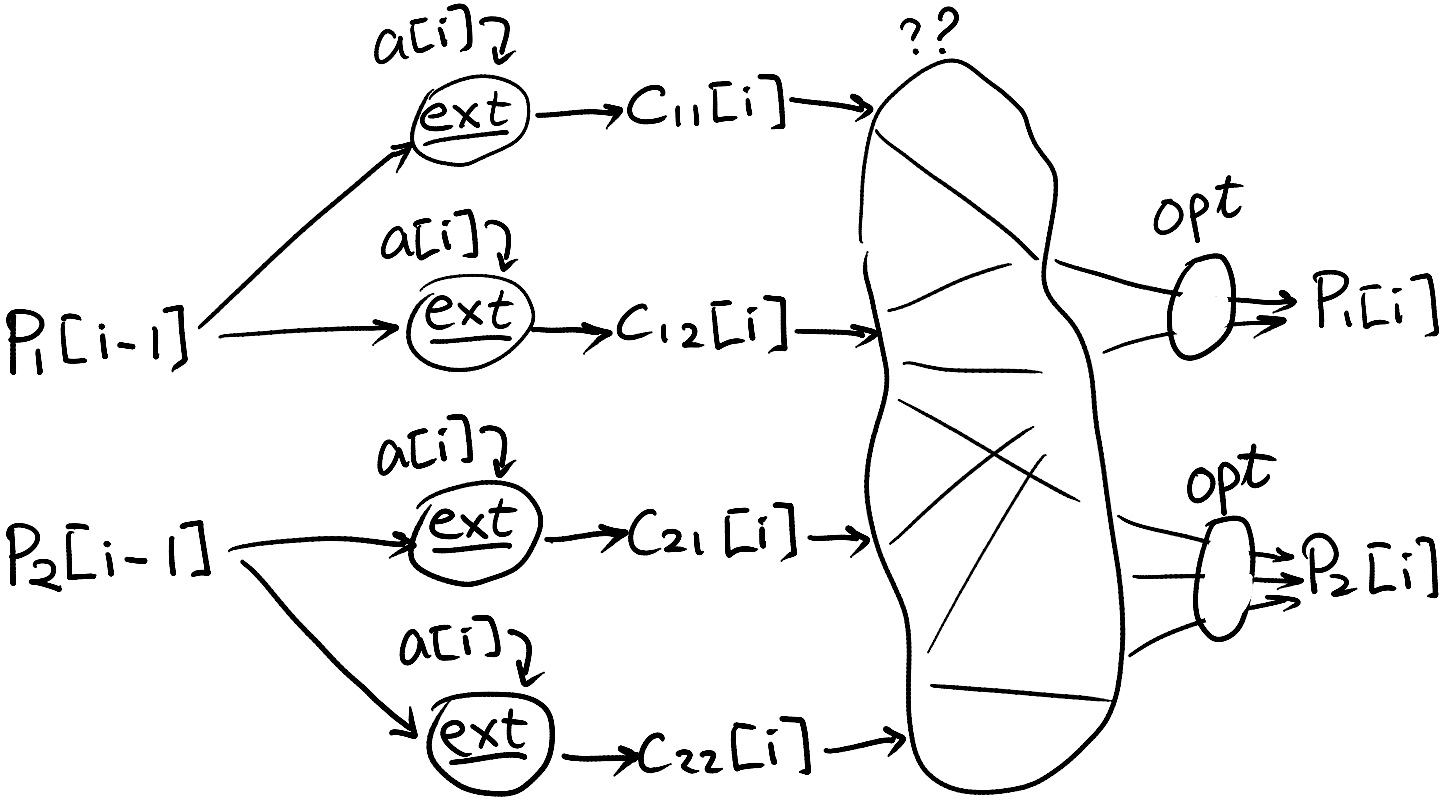
All possible compositions of user provided operators





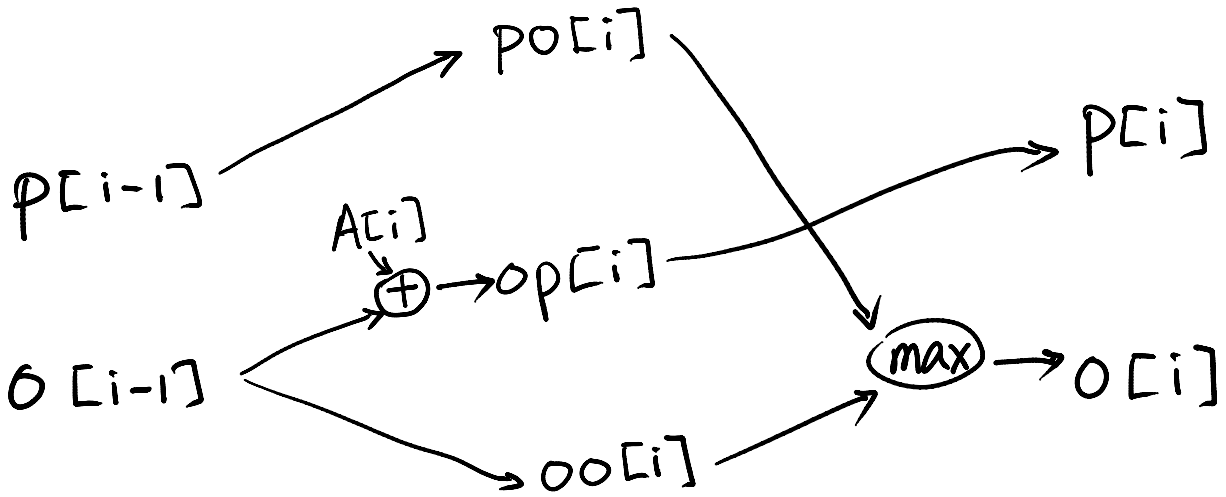
Space Reduction: Optimality

All FORDP recurrences have this syntactic form



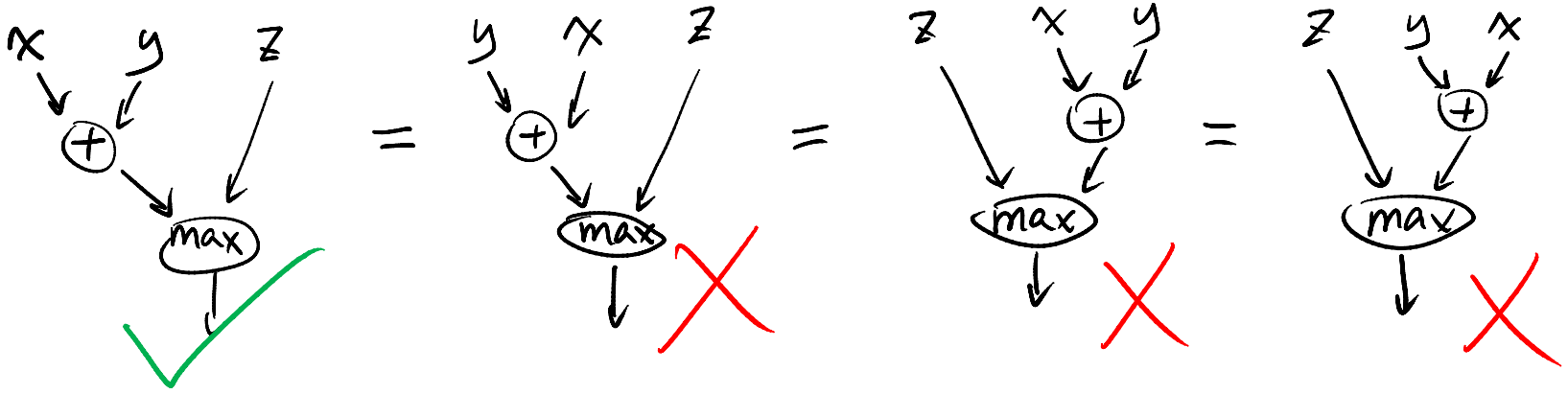
Space Reduction: Optimality

Recurrence for MIS:



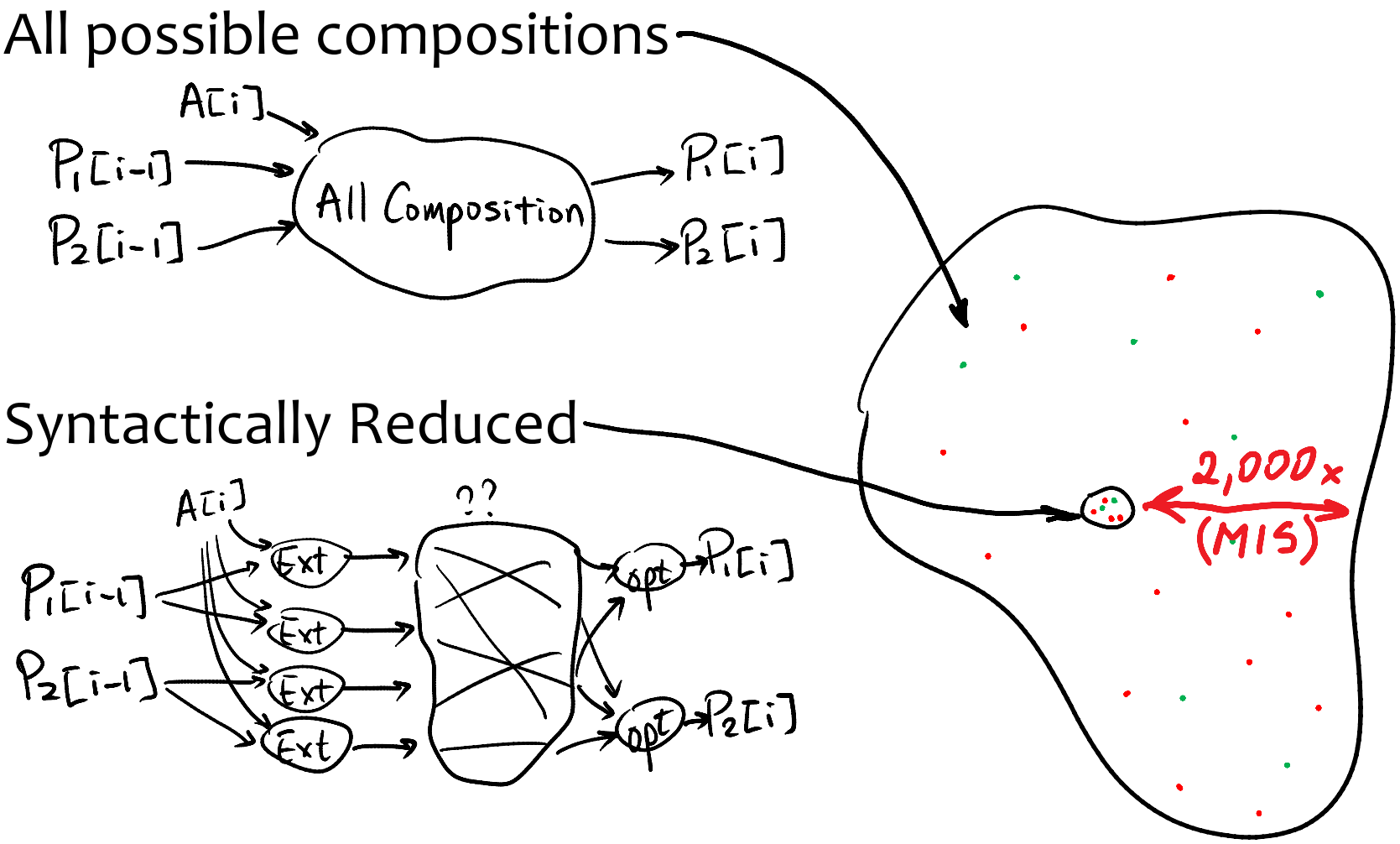
Space Reduction: Symmetry

Many operators are commutative



Pick a canonical representative syntactically

Space Reduction: Recap



DEMO

Benchmarks

Here are some synthesized recurrences

mis:

$$p(i) = o(i-1) + a(i)$$
$$o(i) = \max(p(i-1), o(i-1))$$

assm:

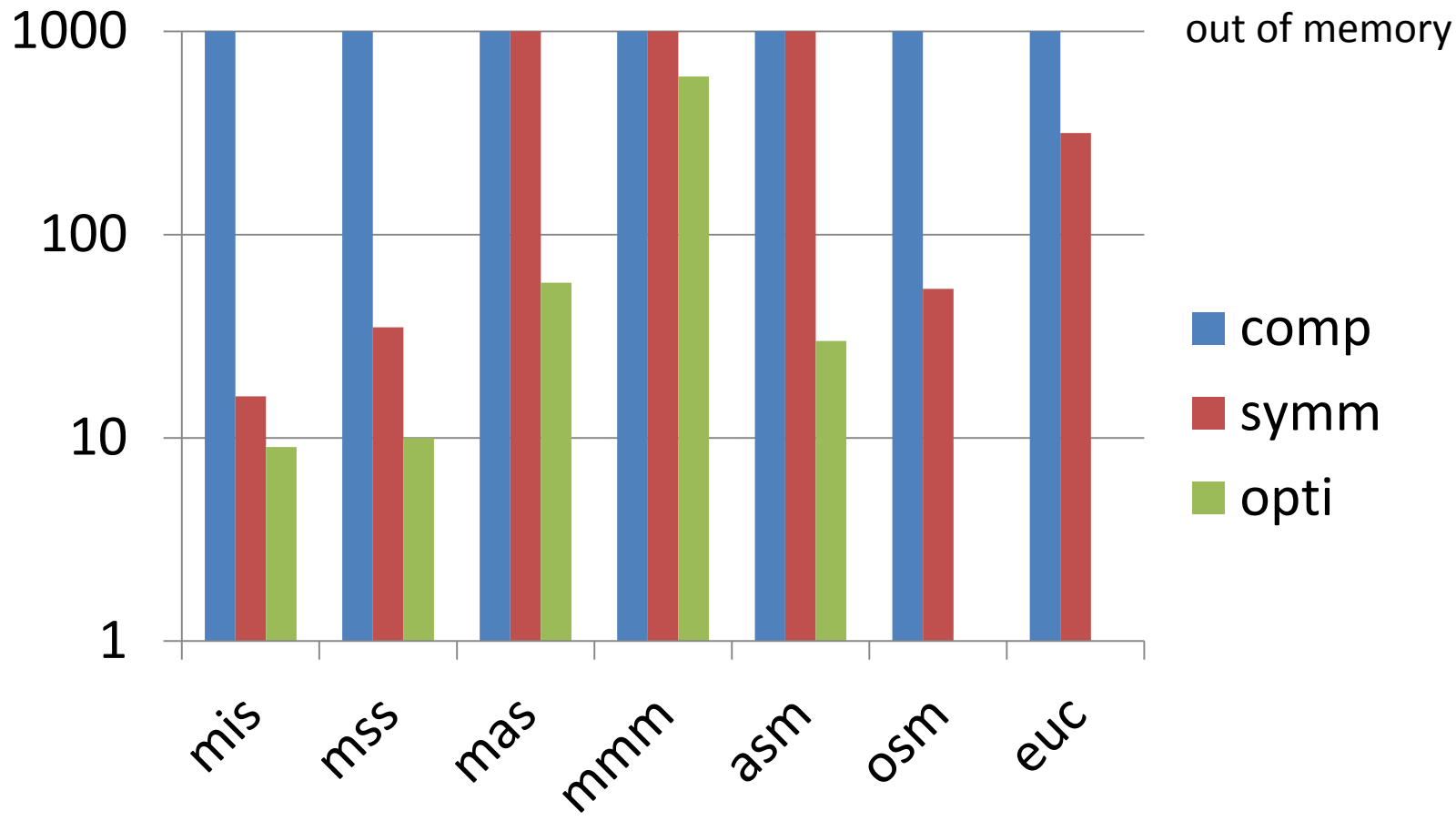
$$l_1(i) = \min(l_1(n-1) + \text{stay}_1(n), l_2(n-1) + \text{switch}_2(n))$$
$$l_2(i) = \min(l_2(n-1) + \text{stay}_2(n), l_1(n-1) + \text{switch}_1(n))$$

extended euclid:

$$P_1(i) = P_2(i-1)$$
$$P_2(i) = P_1(i-1) + P_2(i-1) * \frac{q_1(n)}{q_2(n)}$$

Experiments

Synthesizer solving time, in seconds



Conclusion

It is possible to build a domain-specific synthesizer for FORDPA

Synthesizer developer only find a **syntactic domain structure**

The lessons learned in building the synthesizer may be general

If so, we can build more islands with constraint-based synthesis