Lecture 5
Stochastic and Representation-based Search

Nadia Polikarpova

Logistics

Project topics

- Once you have decided on the topic, put it on the Google sheet next to any of the team members
- If you haven't decided, talk to me

Project proposals

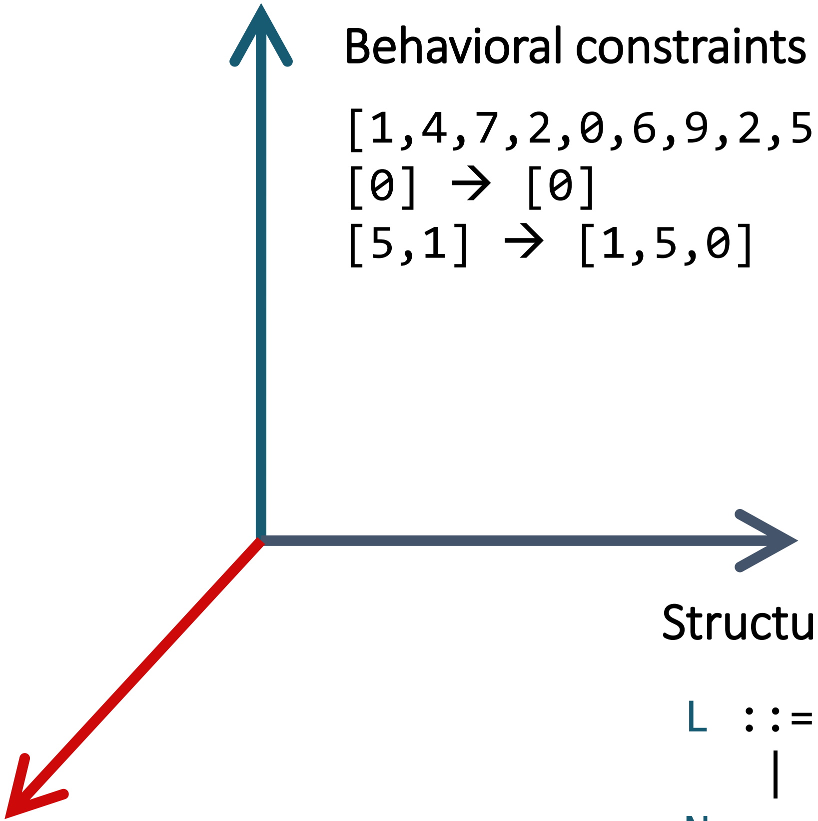
- Due next Friday (Oct 26)
- Should demonstrate that you started working on the project or at least researched the area
- So start early!

Stochastic search

The problem statement

Behavioral constraints = examples

```
[1,4,7,2,0,6,9,2,5]  →  [1,2,4,7,0]
[0] → [0]
[5,1] → [1,5,0]
```



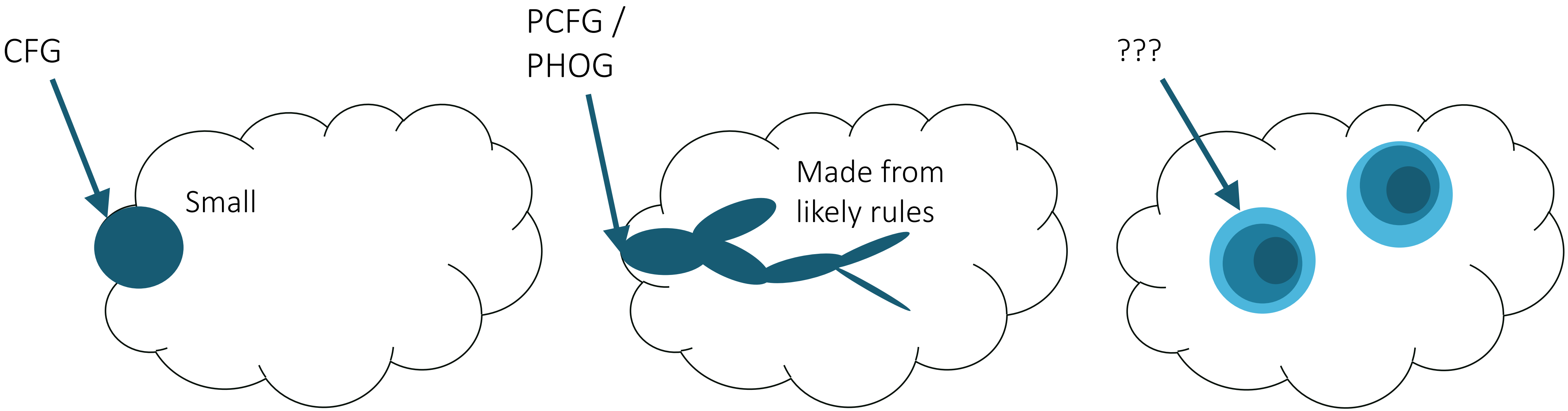
Search strategy?

Enumerative
Stochastic
Representation-based
Constraint-based

Structural constraints = grammar

```
L ::= sort(L)  |  L[N..N]
    |  L + L  |  [N]  |  x
N ::= find(L,N)  |  0
```

Search space



Enumerative search

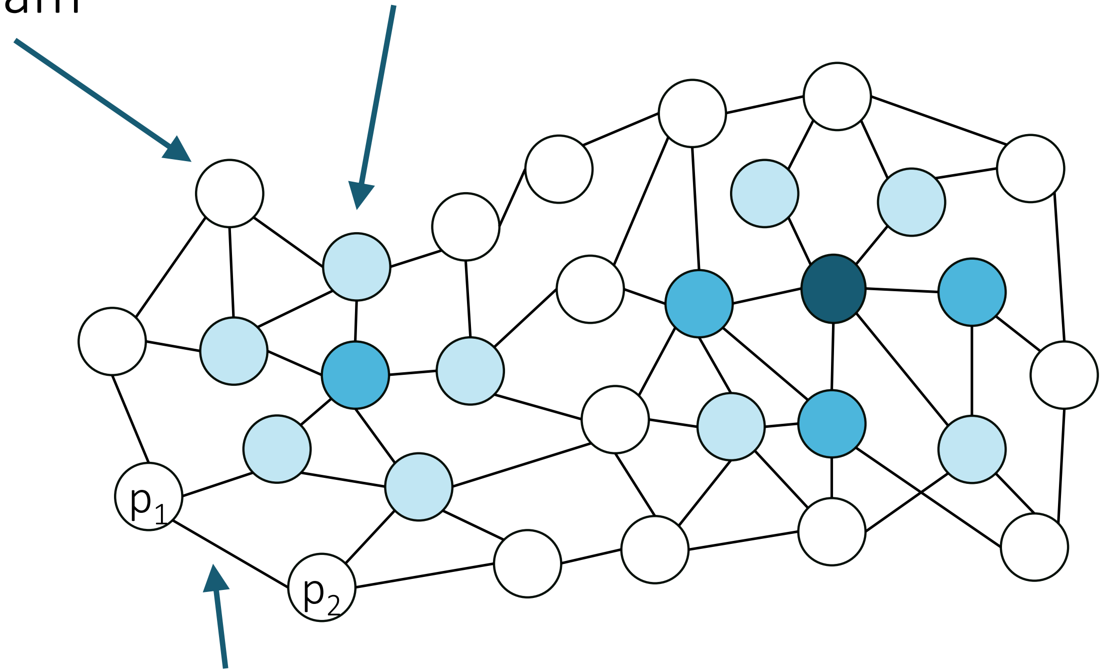
Weighted
enumerative search

Stochastic!

Search by hill climbing

To find the best program:

```
p := random()
while (true) {
  p’ := mutate(p);
  if (cost(p’) < cost(p))
    p := p’;
}
```



Will never get to ● from p_1!

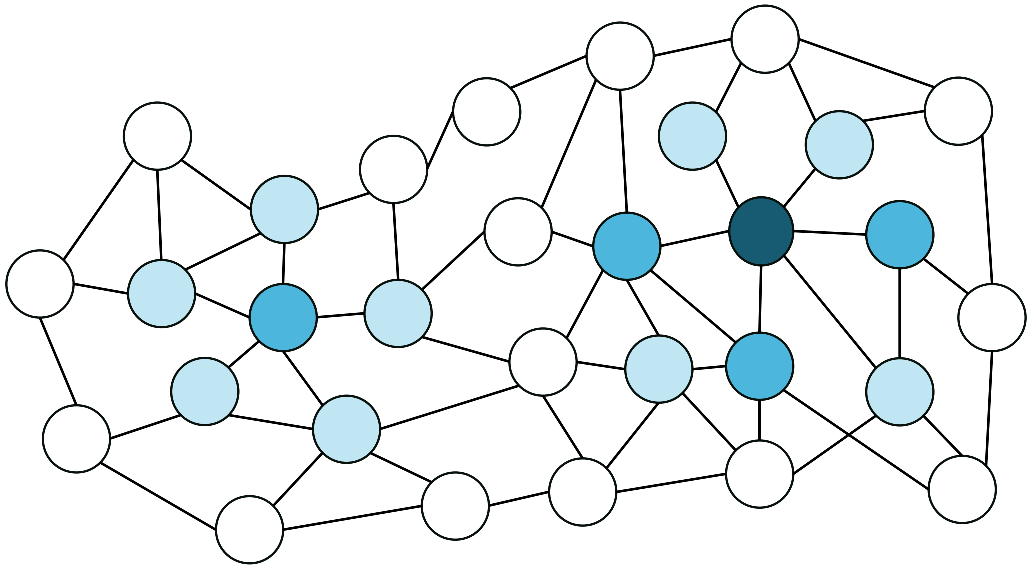
MCMC sampling

Avoid getting stuck in local minima:

```
p := random()
while (true) {
  p’ := mutate(p);
  if (random(A(p -> p’))
    p := p’;
}
```

where

- if p’ is better than p: $A(p \rightarrow p') = 1$
- otherswise: $A(p \rightarrow p')$ decreases with difference in cost between p’ and p



MCMC sampling

Metropolis algorithm:

$$A(p \to p') = \min(1, e^{-\beta(C(p')-C(p))})$$

The theory of Markov chains tells us that in the limit we will be sampling with the probability proportional to

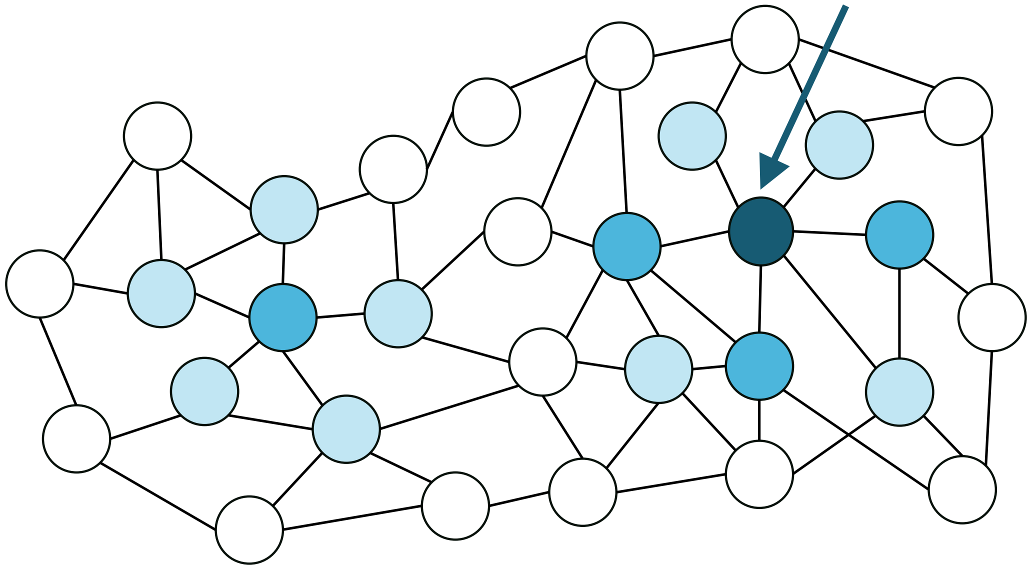
$$e^{-\beta * C(p)}$$

MCMC for superoptimization

[Schkufza, Sharma, Aiken ‘13]

```
.L0:
movq rsi, r9
movl ecx, ecx
shrq 32, rsi
andl 0xffffffff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rdx, r9
imulq rsi, rdx
imulq rsi, rcx
addq rdx, rax
jae .L2
movabsq 0x100000000, rdx
addq rdx, rcx
.L2:
movq rax, rsi
movq rax, rdx
shrq 32, rsi
salq 32, rdx
addq rsi, rcx
addq r9, rdx
adcq 0, rcx
addq r8, rdx
adcq 0, rcx
addq rdi, rdx
adcq 0, rcx
movq rcx, r8
movq rdx, rdi
```

```
.L0:
shlq 32, rcx
movl edx, edx
xorq rdx, rcx
movq rcx, rax
mulq rsi
addq r8, rdi
adcq 0, rdx
addq rdi, rax
adcq 0, rdx
movq rdx, r8
movq rax, rdi
```



Cost function

$$C_s(p) = \mathrm{eq}_s(p) + \mathrm{perf}(p)$$

source program

penalty for wrong results

penalty for being slow

$$\mathrm{eq}_s(p) = \sum_{t \in Tests} \mathrm{reg}_s(p,t) + \mathrm{mem}_s(p,t) + \mathrm{err}(p,t)$$

# of different bits in registers/memory

# of segfaults etc

when $\mathrm{eq}_s(p) = 0$, use a symbolic validator

Cost function

$$C_s(p) = \mathrm{eq}_s(p) + \mathrm{perf}(p)$$

source program

penalty for wrong results

penalty for being slow

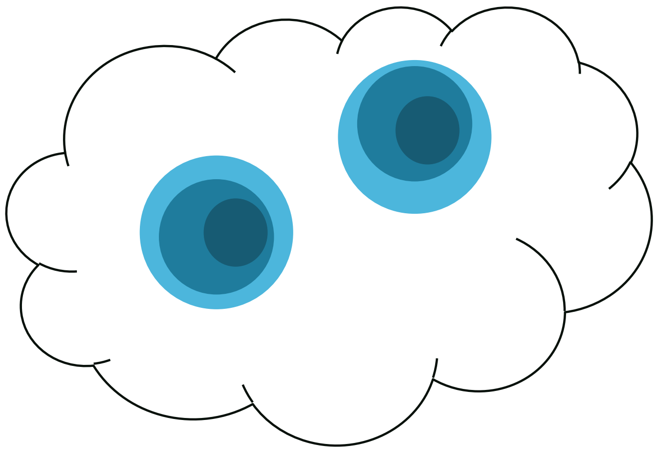
$$\mathrm{perf}(p) = \sum_{i \in \mathrm{instr}(p)} \mathrm{latency}(i)$$

Stochastic search: discussion

Hill climbing can explore larger spaces

Limitations?

- only applicable when there is a cost function that faithfully approximates correctness
- Counterexample: round to next power of two



Representation-based search

The problem statement

Behavioral constraints = examples

```
[1,4,7,2,0,6,9,2,5]  →  [1,2,4,7,0]
[0] → [0]
[5,1] → [1,5,0]
```

Search strategy?

Enumerative

Stochastic

Representation-based

Constraint-based

Structural constraints = grammar

```
L ::= sort(L)  |  L[N..N]
    |  L + L  |  [N]  |  x
N ::= find(L,N)  |  0
```

Representation-based search

Idea: pick a data structure that can succinctly represent a set of programs consistent with a spec

- called a **version space**

Operations on version spaces:

- `learn <i, o> → VS`
- $VS_1 \cap VS_2 \rightarrow VS$
- `pick VS → program`

Sounds too good to be true?

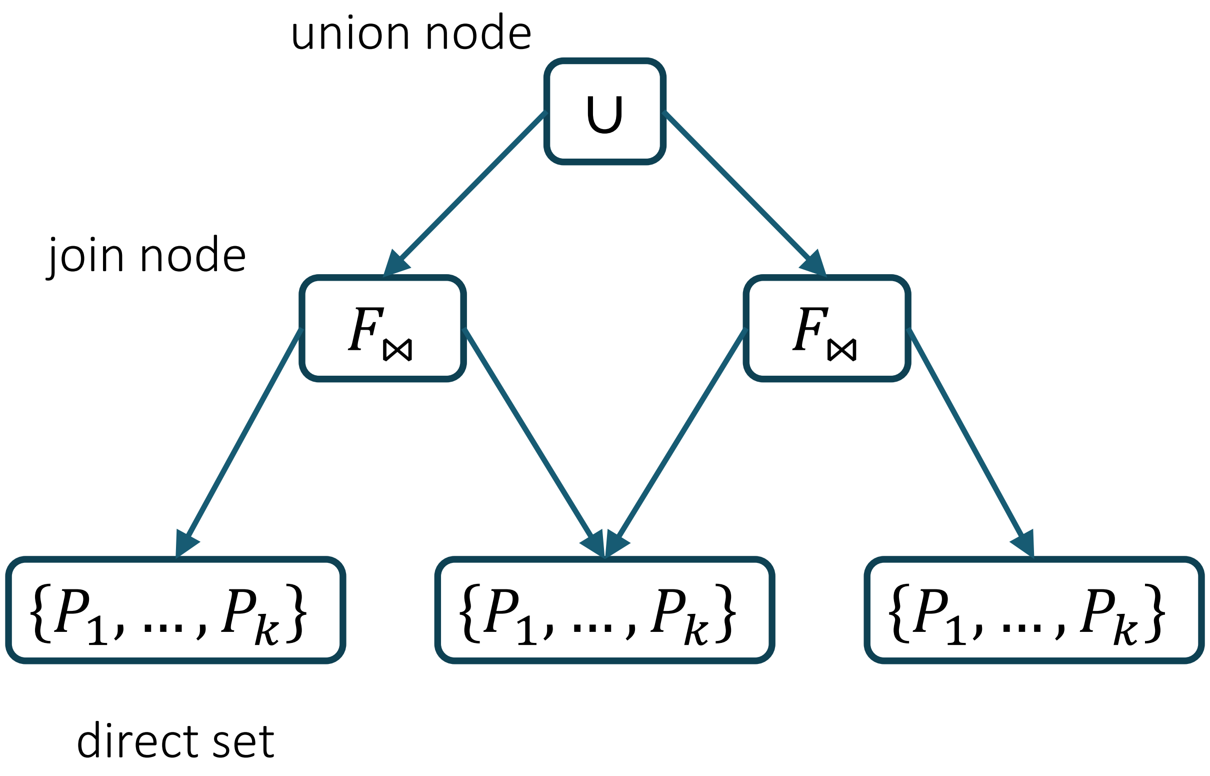
- Let's see when it works

Representations?

Version Space Algebras

Finite Tree Automata

Version Space Algebra



Volume of a VSA
(the number of nodes) $V(VSA)$

Size of a VSA
(the number of programs) $|VSA|$

$$V(VSA) = O(\log|VSA|)$$

Example: FlashFill

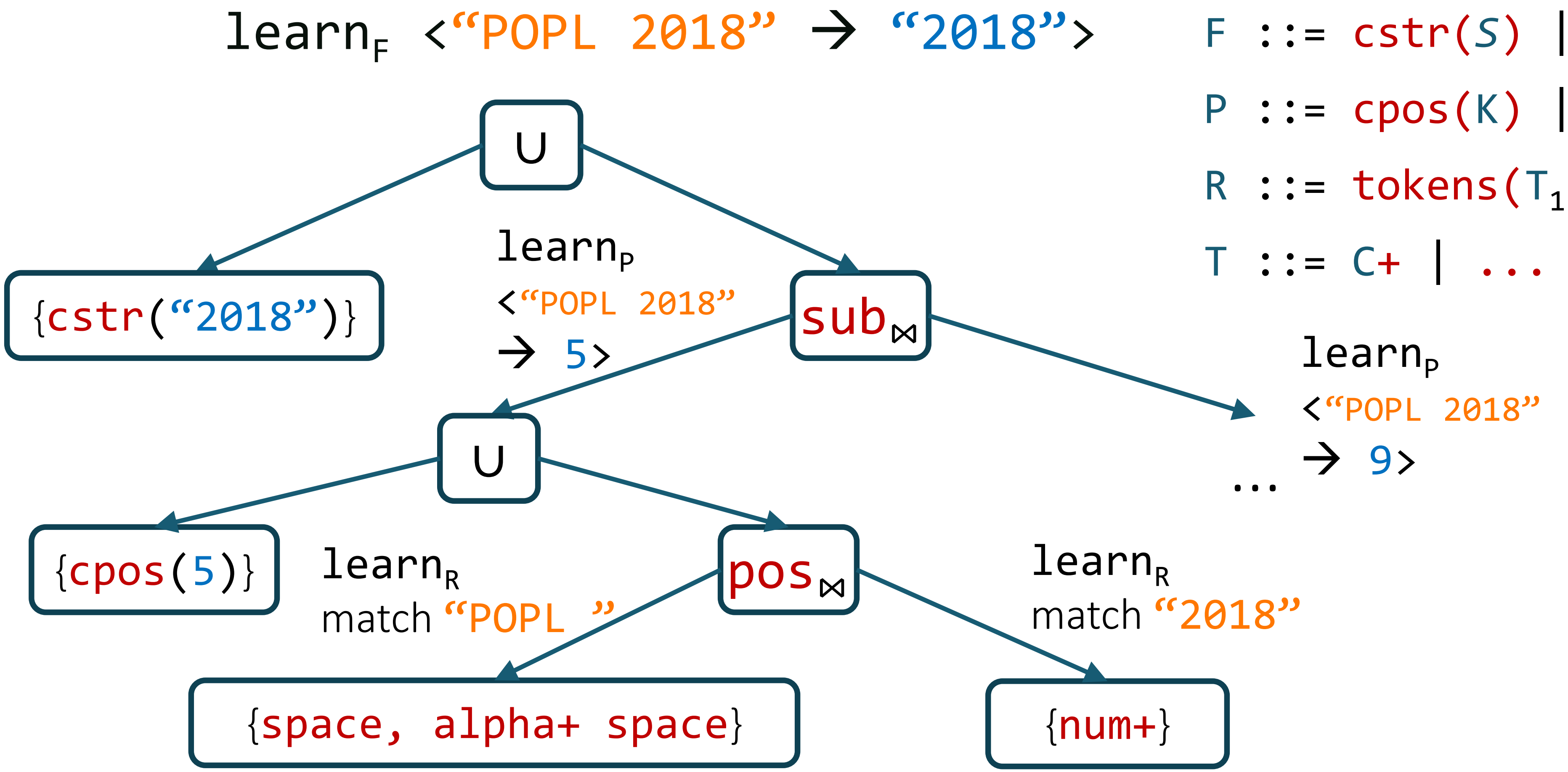
[Gulwani '11]

Simplified grammar:

`E ::= F	concat(F, E)`	"Trace" expression
`F ::= cstr(S)	sub(P, P)`	Atomic expression
`P ::= cpos(K)	pos(R, R)`	Position expression
`R ::= tokens(T_1, ..., T_n)`	Regular expression
`T ::= C+	...`	Token expression

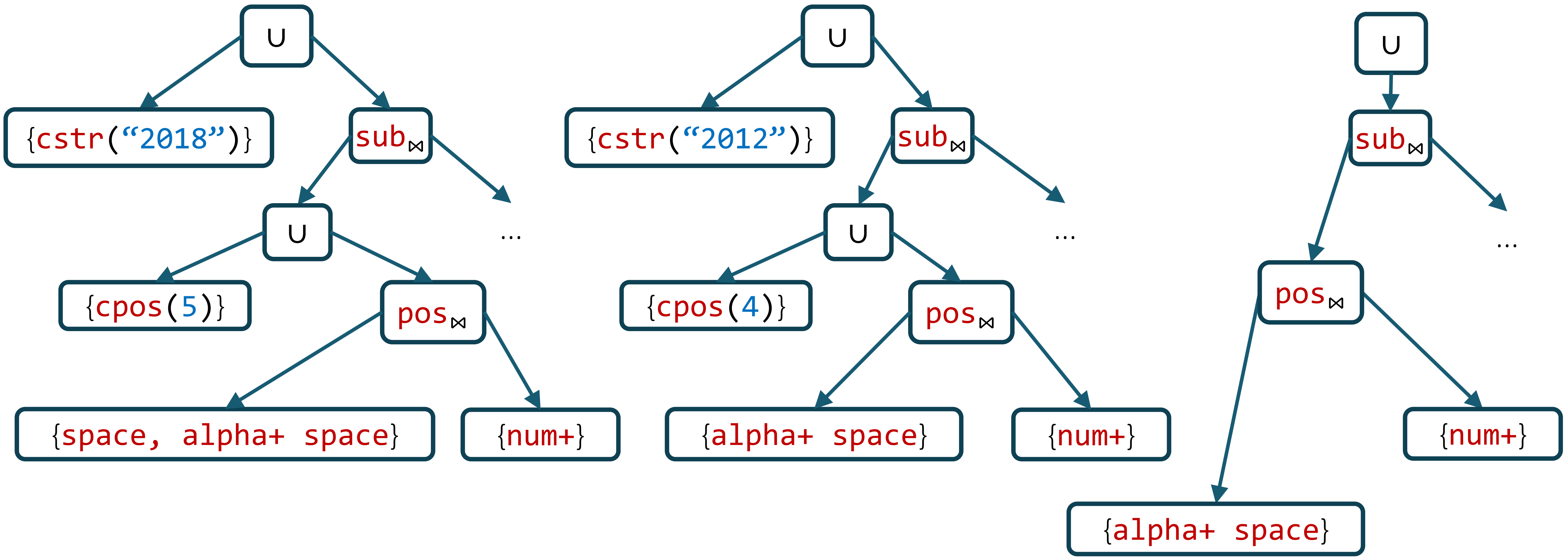
Learning atomic expressions

F ::= cstr(S) | sub(P₁, P₂)



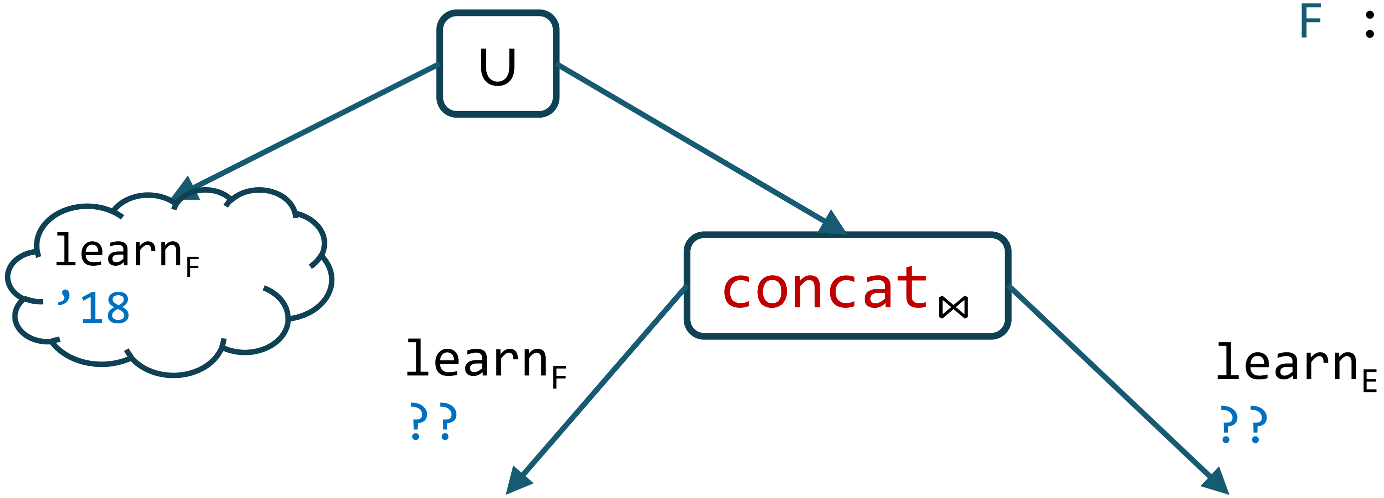
Intersection

“POPL 2018” → “2018” “ FM 2012” → “2012”



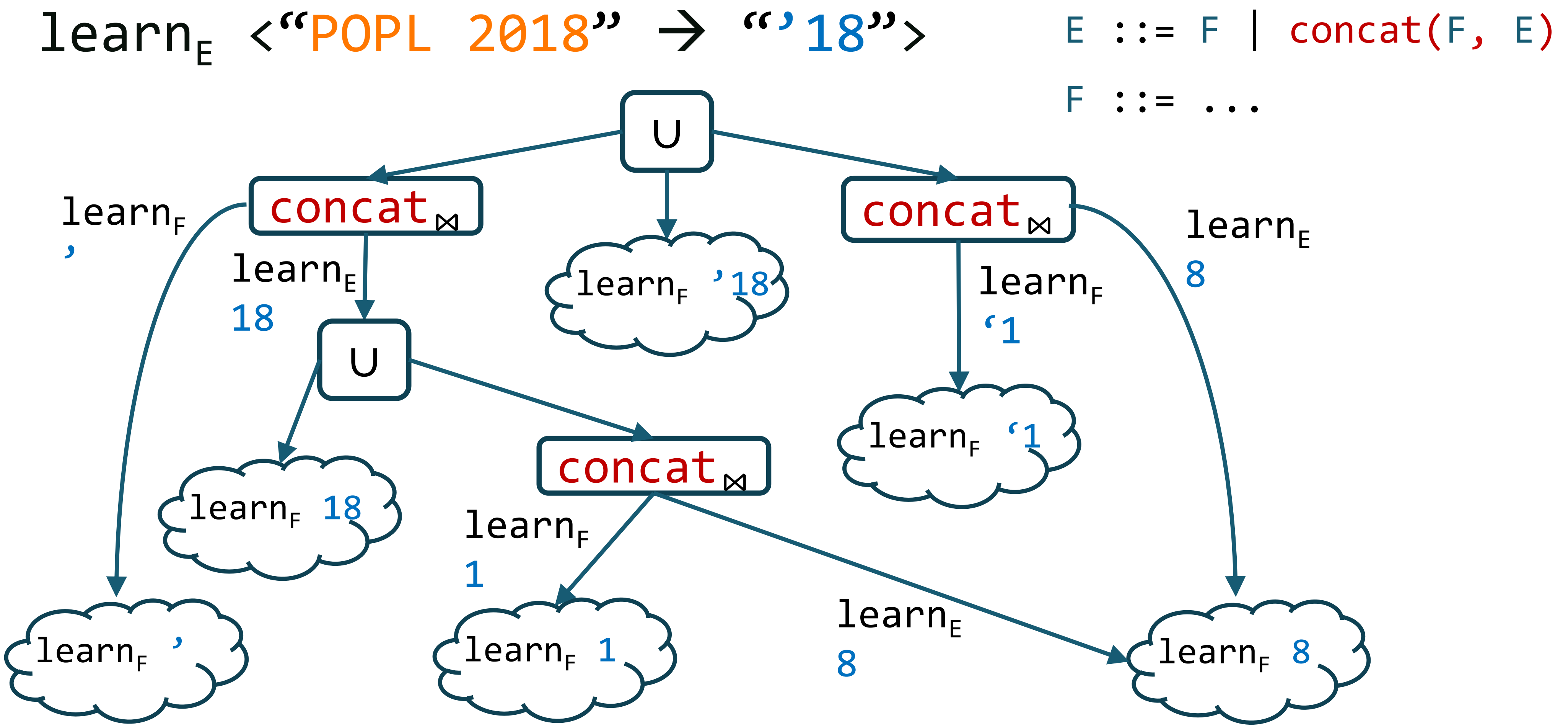
Learning trace expressions

learn$_E$ <"POPL 2018" → "'18">




```
E ::= F | concat(F, E)
F ::= ...
```

Learning trace expressions



Discussion

Why could we build a finite representation of all expressions?

- Could we do it for this language?

$$\begin{aligned} E &::= F + F \\ F &::= K \mid x \end{aligned}$$

$K \in \mathbb{Z}$ + is integer addition

- What about this language?

$$\begin{aligned} E &::= F \mid F + E \\ F &::= K \mid x \end{aligned}$$

$K \in [0,9]$ + is addition mod 10

Discussion

Why could we build a *compact* representation of all expressions?

- Could we do this for this language?

$$E ::= F \;\&\; F$$

$$F ::= K \mid x$$

K is a 32-bit word, $\&$ is bit-and

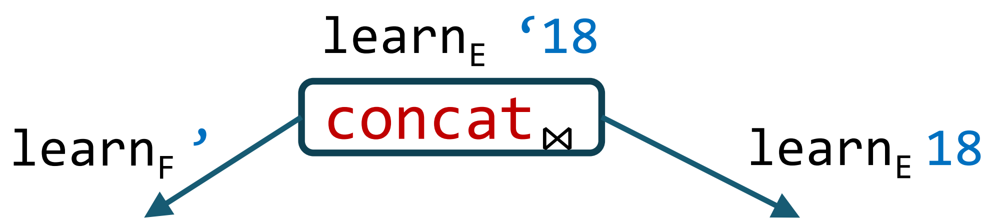
VSA: DSL restrictions

Every operator has a small, easily computable inverse

- Example when an inverse is small but hard to compute?

Every recursive rule generates a strictly smaller subproblem

```
E ::= F | concat(F, E)
```



- Otherwise, limit depth and “unroll” the grammar

PROSE

[Polozov, Gulwani '15]

Ranking function

```
r :: Program -> Int
```

Grammar

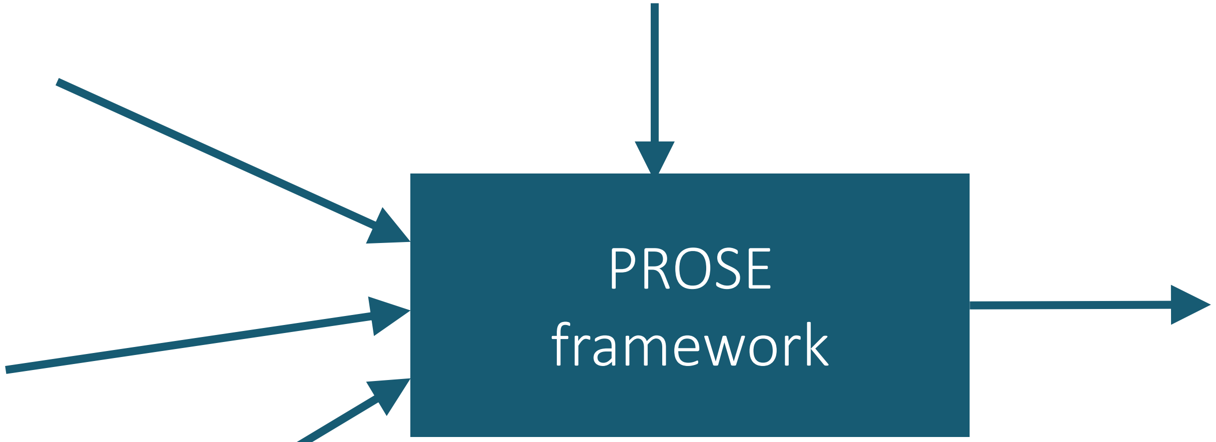
```
A ::= f(B, C) | ...
B ::= g(D, E) | ...
```

Semantics

```
f :: (B, C) -> A
f (b, c) = ...
```

Inverse semantics

```
f⁻¹ :: A -> {(B, C)}
f⁻¹ a = ...
```



VSA learning algorithm

https://microsoft.github.io/prose/

Representations?

Version Space Algebras

Finite Tree Automata

Example

Grammar

```
N ::= id(V) | N + T | N * T
T ::= 2 | 3
V ::= x
```

Spec

1 → 9

Finite Tree Automata

$\langle A, \mathbb{Z} \rangle$

$A \in \{\mathsf{N}, \mathsf{T}, \mathsf{X}\}$

$\{\langle \mathsf{N}, 9 \rangle\}$

states

final states

$$\mathcal{A} = \langle Q, F, Q_f, \Delta \rangle$$

alphabet

transitions

`id`, `+`, `*`

$f(q_1, \ldots, q_n) \rightarrow q$

```
+(<N,1>,<T,2>) → <N,3>
...
```

Finite Tree Automata

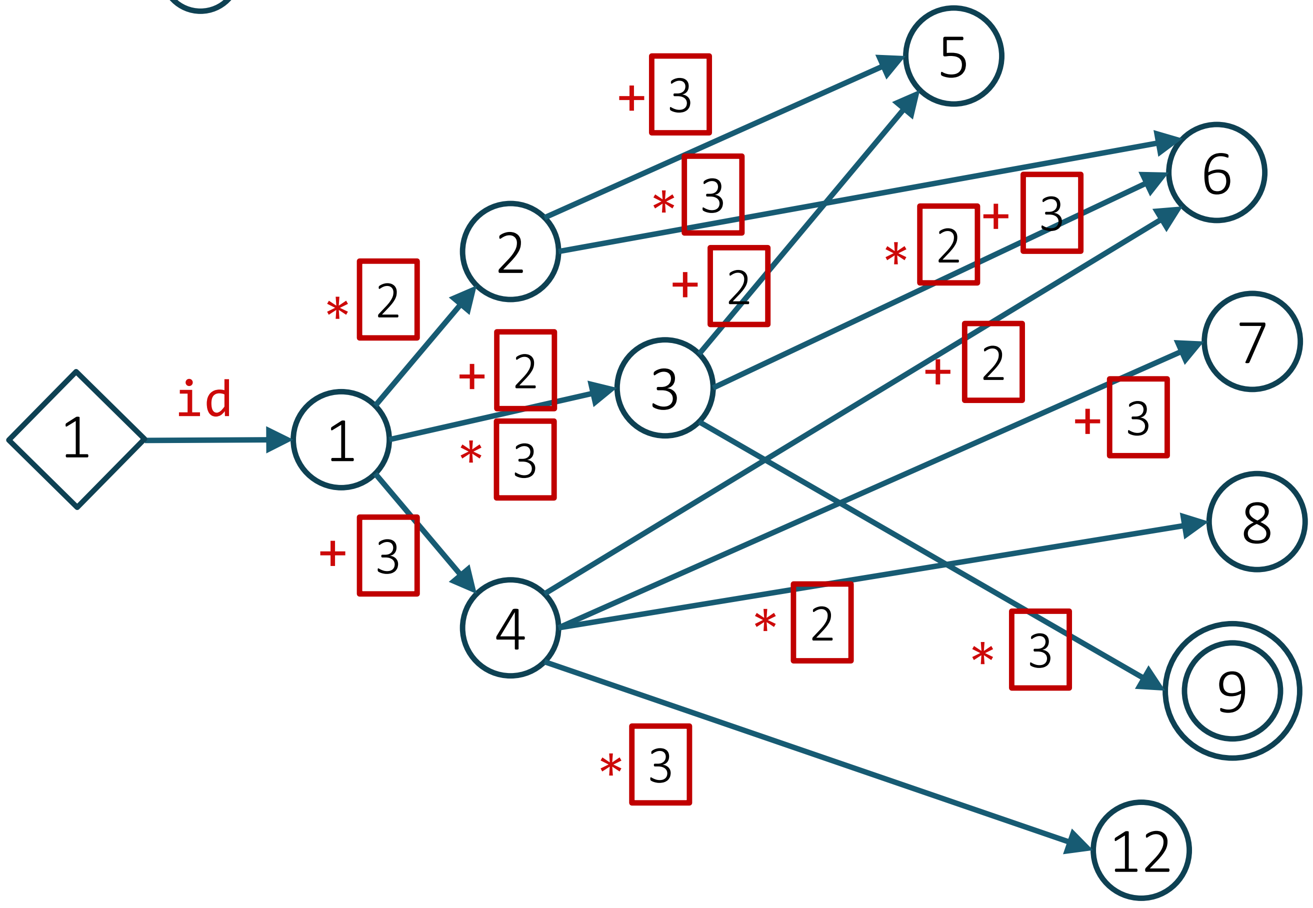
[Wang, Dillig, Singh ‘17]

N ::= id(V) | N + T | N * T

T ::= 2 | 3

V ::= x

1 → 9



Discussion

Representation-based vs enumerative

- Enumerative unfolds the search space in time, while representation-based stores it in memory
- Benefits / limitations?

FTA ~ bottom-up

- with observational equivalence

VSA ~ top-down

- with top-down propagation

Next

Behavioral constraints = examples

```
[1,4,7,2,0,6,9,2,5]  →  [1,2,4,7,0]
[0] → [0]
[5,1] → [1,5,0]
```

Search strategy?

Enumerative
Stochastic
Representation-based
Constraint-based

Structural constraints = grammar

```
L ::= sort(L)  |  L[N..N]
    |  L + L  |  [N]  |  x
N ::= find(L,N)  |  0
```