Lecture 5
Representation-based Search

Nadia Polikarpova

This week

Topics:

- Representation-based search
- Stochastic search

Paper: Rishabh Singh: BlinkFill: Semisupervised Programming By Example for Syntactic String Transformations. VLDB'16

Projects:

- Proposals due Friday
- Should demonstrate that you started working on the project or at least researched the area
- 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 after class or in OH

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]
```

Structural constraints = grammar

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

Search strategy?

Enumerative
Representation-based
Stochastic
Constraint-based

Representation-based search

Idea:

1. build a graph that represents the search space
2. search in that graph (or not)

Tradeoff: easy to build vs easy to search

Representations

Version Space Algebra (VSA)

Finite Tree Automaton (FTA)

Type Transition Net (TTN)

Representations

Version Space Algebra (VSA)

Finite Tree Automaton (FTA)

Type Transition Net (TTN)

Version Space Algebra

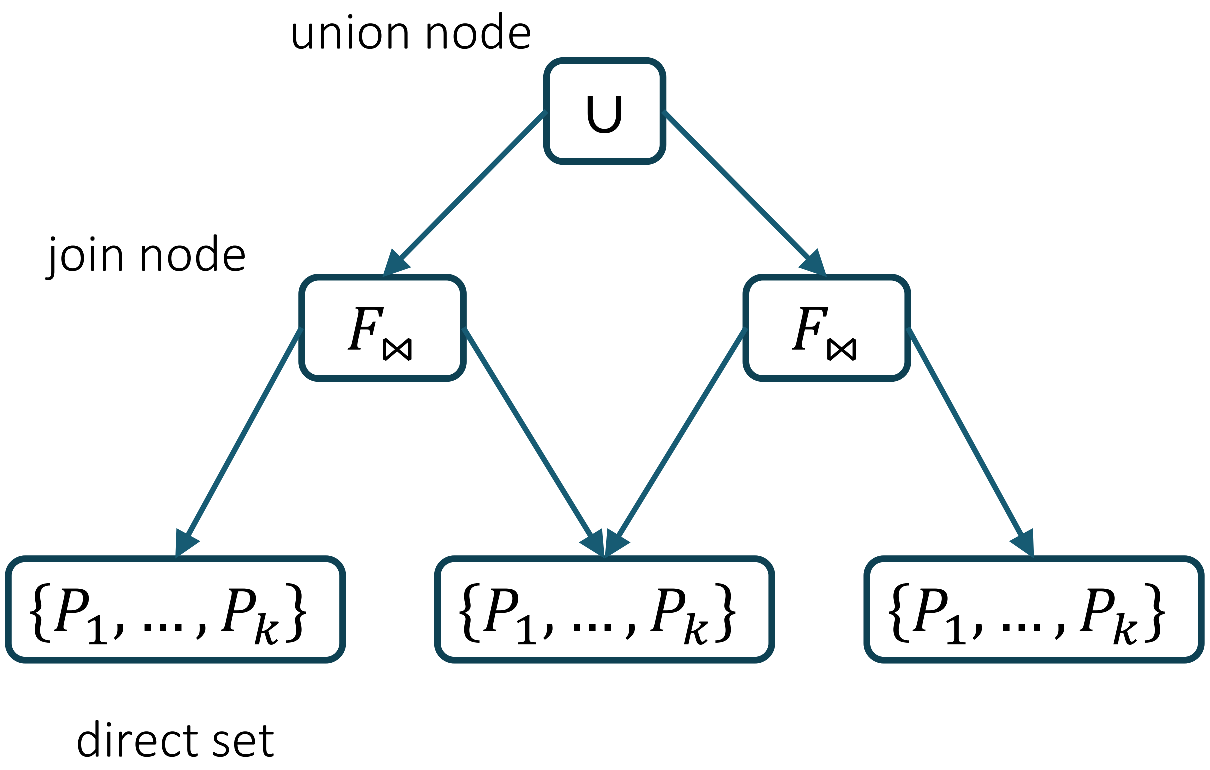
Idea: build a data structure that succinctly represents the set of programs consistent with a spec

- called a **version space**

Operations on version spaces:

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

Version Space Algebra



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

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

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

Version Space Algebra: history

Mitchell: *Generalization as search*. AI 1982

Lau, Domingos, Weld. *Version space algebra and its application to programming by example*. ICML 2000

Gulwani: *Automating string processing in spreadsheets using input-output examples.* POPL 2011.

- BlinkFill, FlashExtract, FlashRelate, …
- generalized in the PROSE framework

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 | C+	Token expression

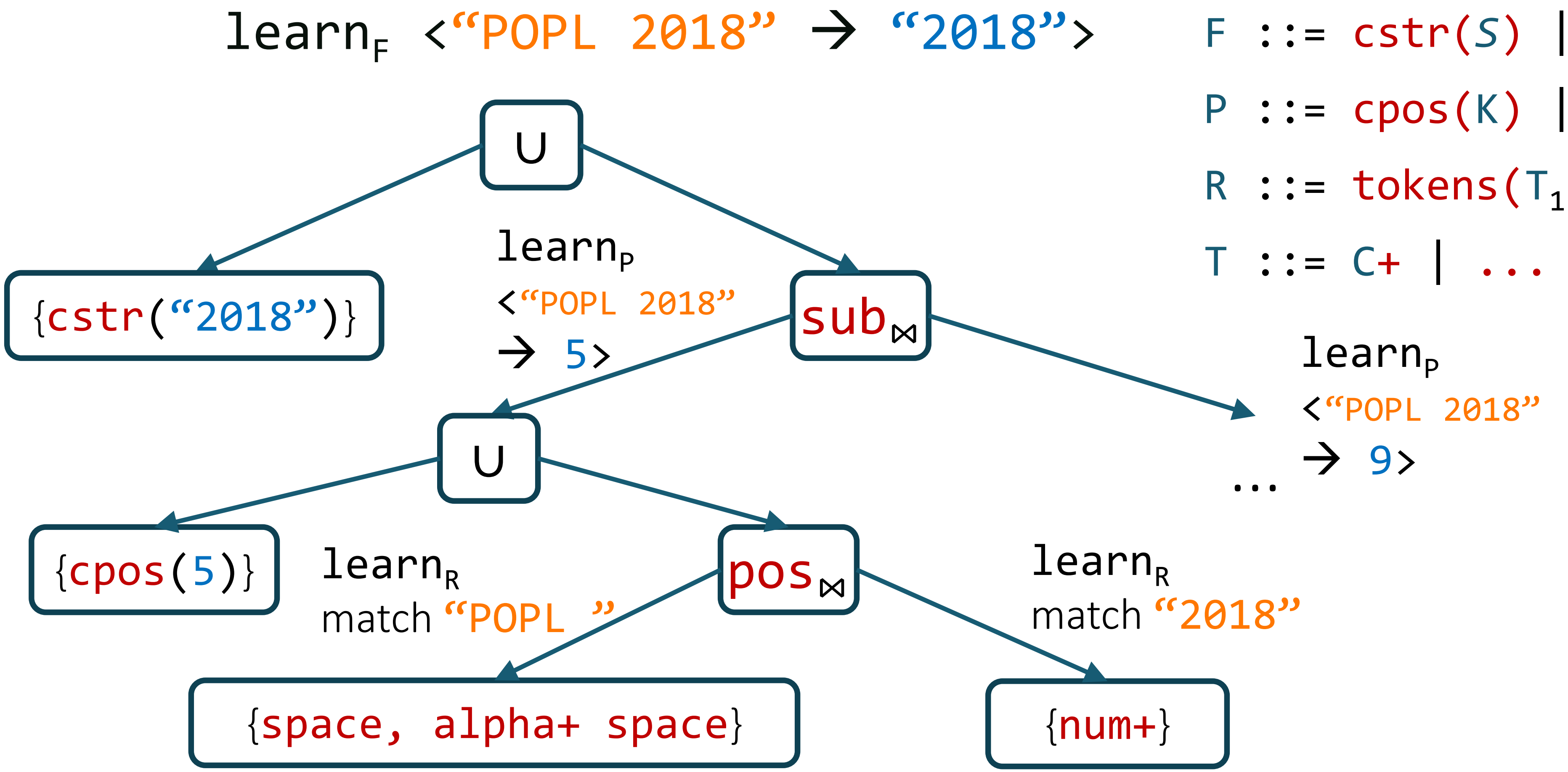
FlashFill: example

“Hello POPL 2020” → “POPL’2020”
“Goodbye PLDI 2019” → “PLDI’2019”

```
E ::= F | concat(F, E)
F ::= cstr(S) | sub(P, P)
P ::= cpos(K) | pos(R, R)
R ::= tokens(T1, ..., Tn)
T ::= C | C+
```

```
concat(
  sub(pos(ws, C), pos(C, ws)),
  concat(
    cstr(“’”),
    sub(pos(ws, d), pos(d, $))))
```

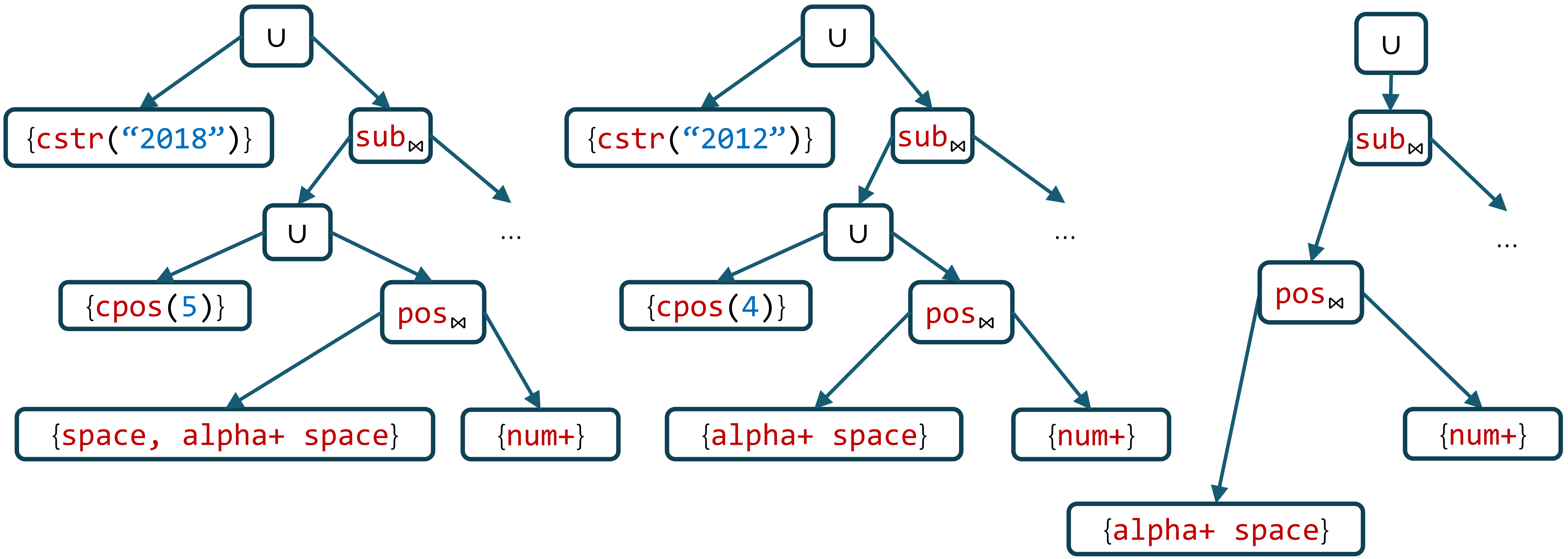
Learning atomic expressions




```
F ::= cstr(S) | sub(P1, P2)
P ::= cpos(K) | pos(R1, R2)
R ::= tokens(T1, ..., Tn)
T ::= C+ | ...
```

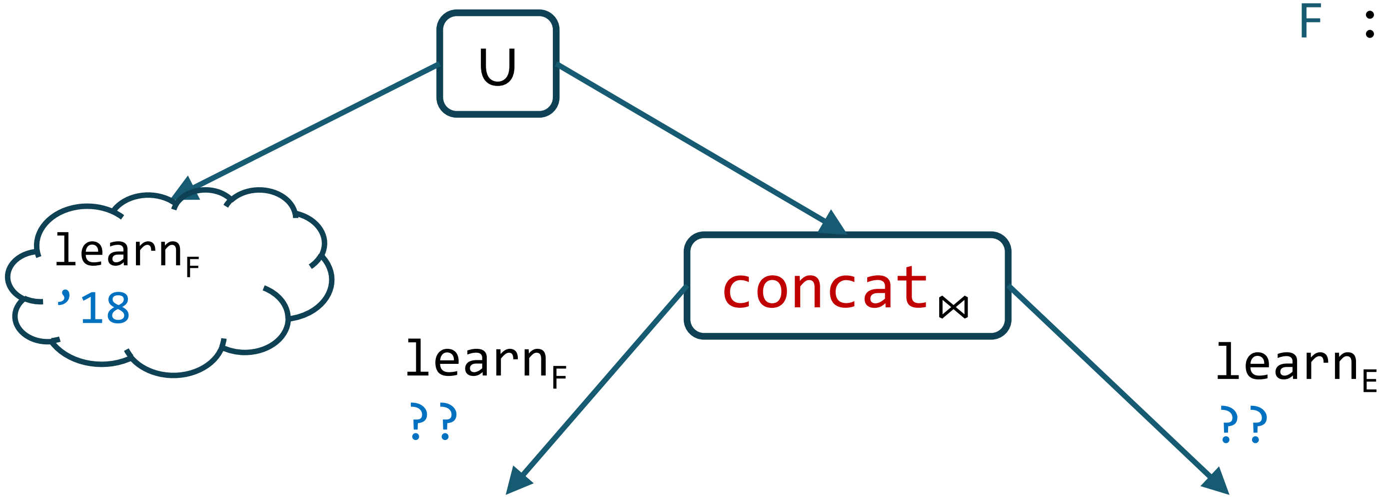
Intersection

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



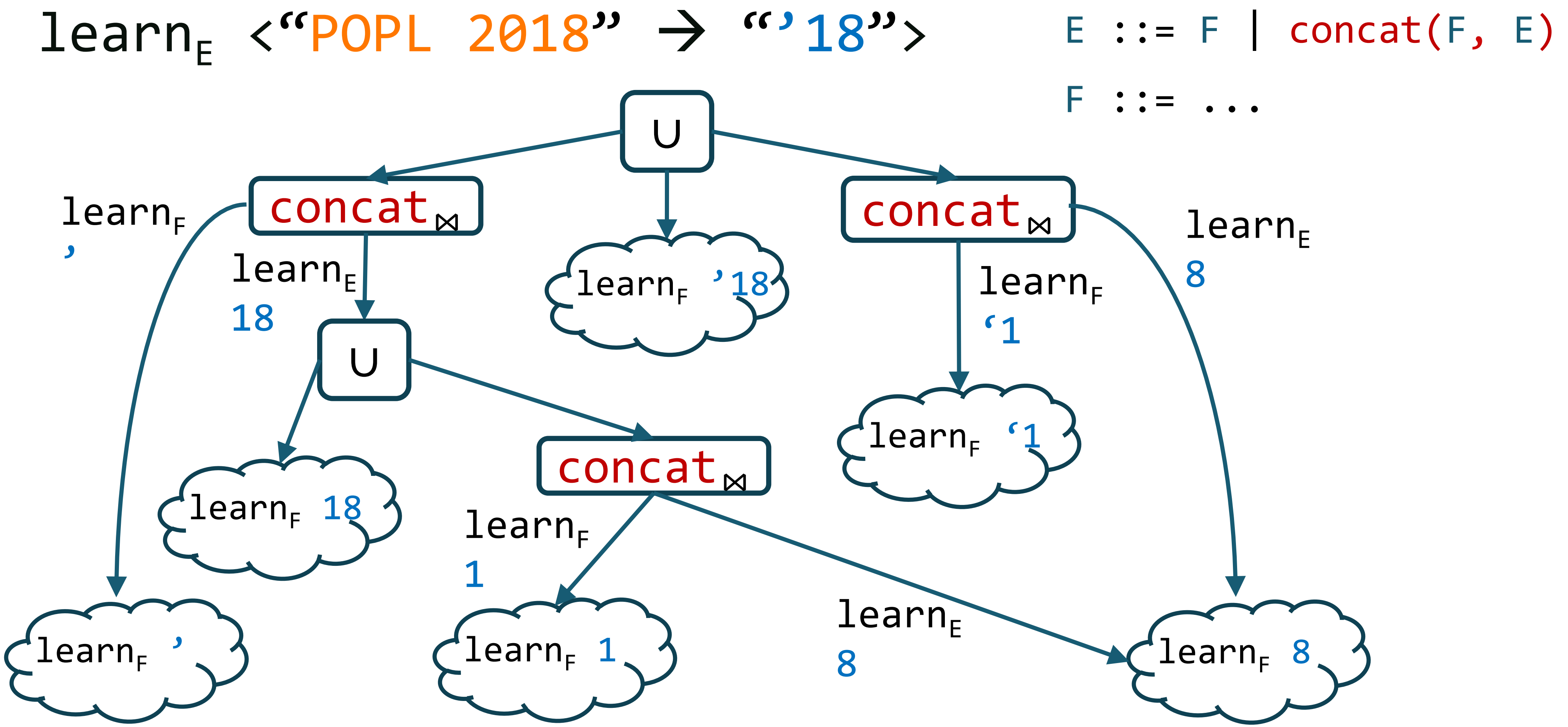
Learning trace expressions

$\texttt{learn}_E$ <“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?

$$E ::= F + F \qquad F ::= K \mid x$$

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

- What about this language?

$$E ::= F \mid F + E \qquad F ::= K \mid x$$

$K \in [0,9]$ $\quad$ + 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  |  x
```

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

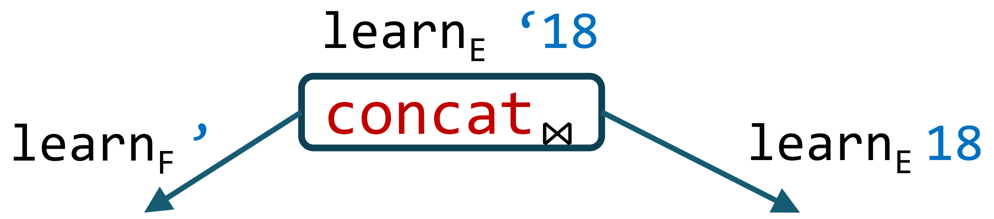
VSA: DSL restrictions

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 = ...`

PROSE framework

VSA learning algorithm

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

Representations

Version Space Algebra (VSA)

Finite Tree Automaton (FTA)

Type Transition Net (TTN)

Example

Grammar

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

Spec

$1 \rightarrow 9$

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

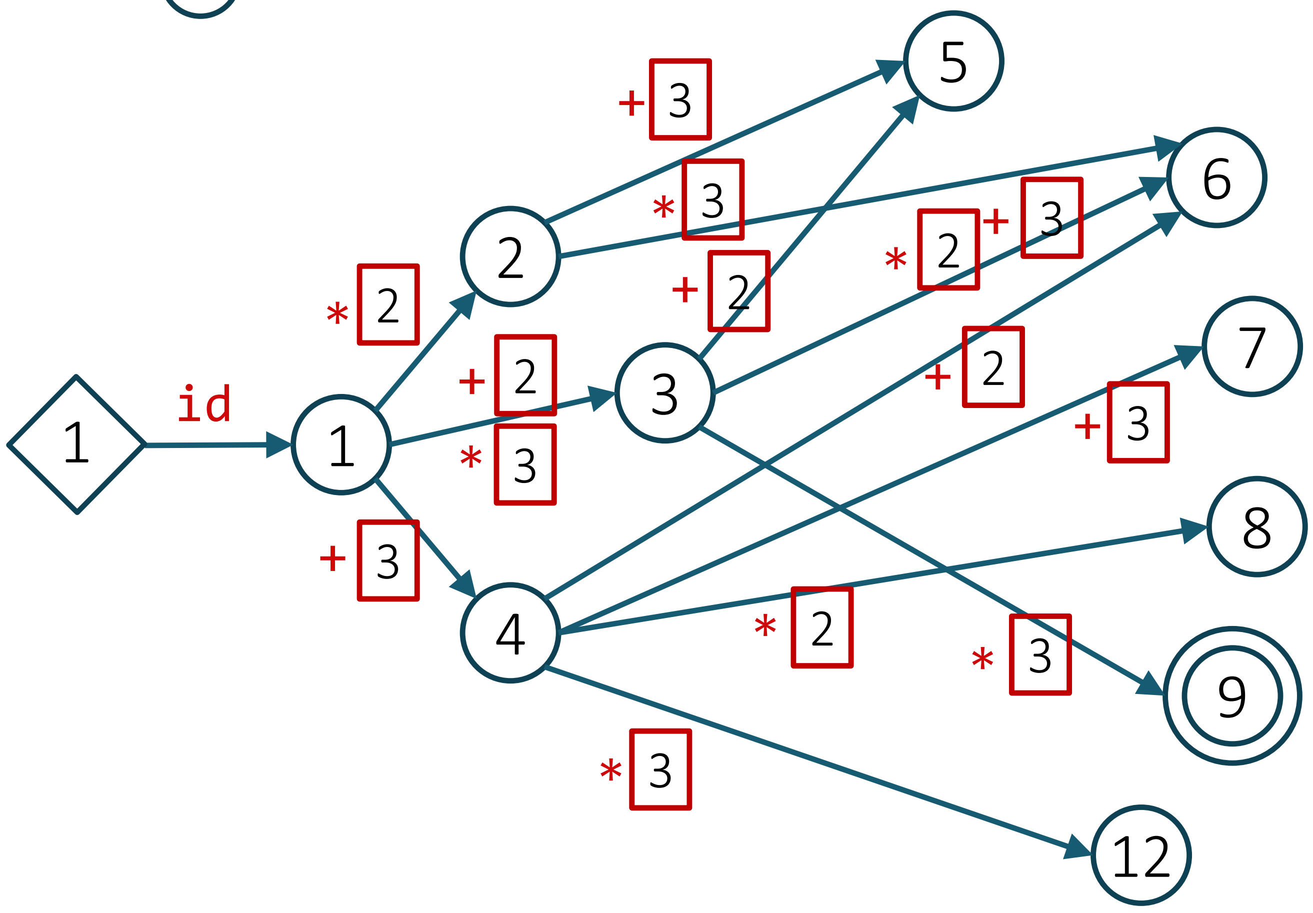
[Wang, Dillig, Singh OOPSLA'17]

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

T ::= 2 | 3

V ::= x

1 → 9



Abstract FTA

Challenge: FTA still has too many states

Idea:

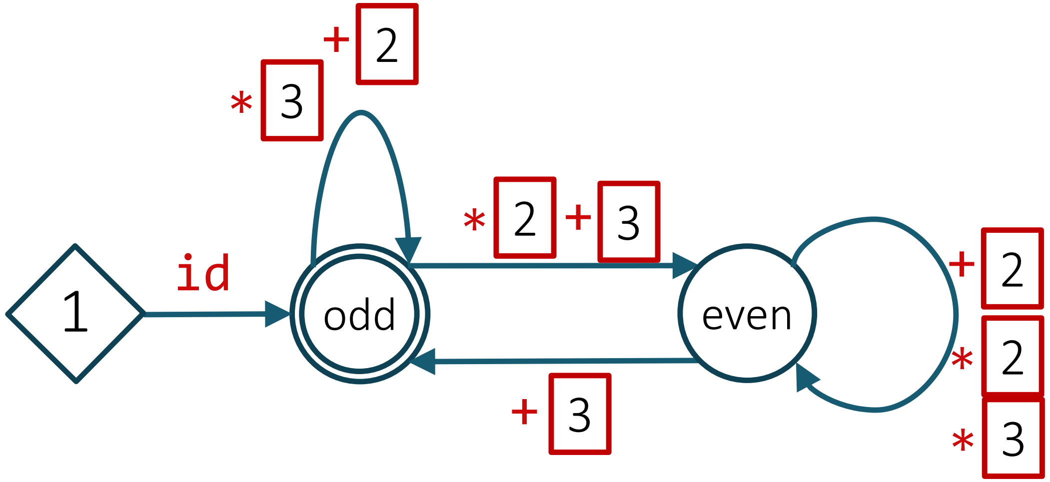
- instead of one state = one value
- we can do one state = set of values (= abstract value)

Abstract FTA

[Wang, Dillig, Singh POPL'18]

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

1 → 9



In the paper:

- different abstractions
- refining the abstractions to eliminate spurious paths

Representations

Version Space Algebra (VSA)

Finite Tree Automaton (FTA)

Type Transition Net (TTN)

Type-Transitions Nets

Context: Component-based synthesis

- given a library of components
- and a query: type + examples?
- synthesize composition of components

Idea: Build a compact graph of the search space using

- types as nodes
- components as transitions

TTN: History

Mandelin, Xu, Bodik, Kimelman: *Jungloid mining: helping to navigate the API jungle.* PLDI’05

Gvero, Kuncak, Kuraj, Piskac: *Complete completion using types and weights.* PLDI’13

Feng, Martins, Wang, Dillig, Reps: *Component-based synthesis for complex APIs.* POPL’17

Guo, James, Justo, Zhou, Wang, Jhala, Polikarpova: *Synthesis by type-Guided Abstraction Refinement.* POPL’20

TTN: History

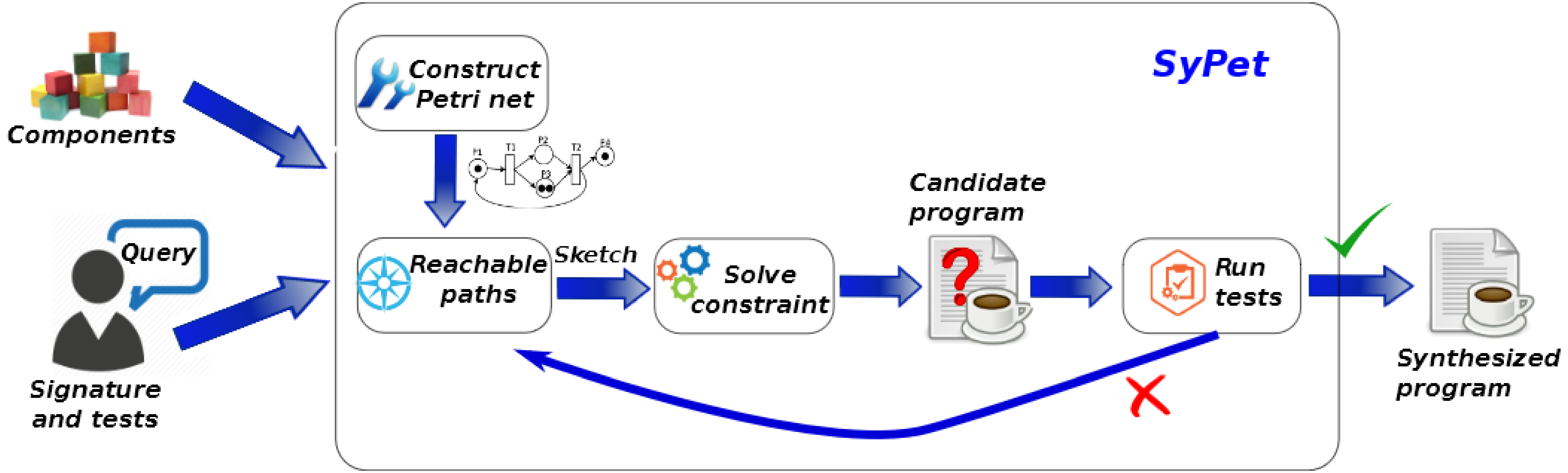
Mandelin, Xu, Bodik, Kimelman: *Jungloid mining: helping to navigate the API jungle.* PLDI’05

Gvero, Kuncak, Kuraj, Piskac: *Complete completion using types and weights.* PLDI’13

Feng, Martins, Wang, Dillig, Reps: *Component-based synthesis for complex APIs.* POPL’17

Guo, James, Justo, Zhou, Wang, Jhala, Polikarpova: *Synthesis by type-Guided Abstraction Refinement.* POPL’20

SyPet: workflow



SyPet: example

Signature

```
Area rotate(Area obj, Point2D pt, double angle)
{ ?? }
```

Test

```
public void test1() {
  Area a1 = new Area(new Rectangle(0, 0, 10, 2));
  Area a2 = new Area(new Rectangle(-2, 0, 2, 10));
  Point2D p = new Point2D.Double(0, 0);
  assertTrue(a2.equals(rotate(a1, p, Math.PI/2)));
}
```

Components

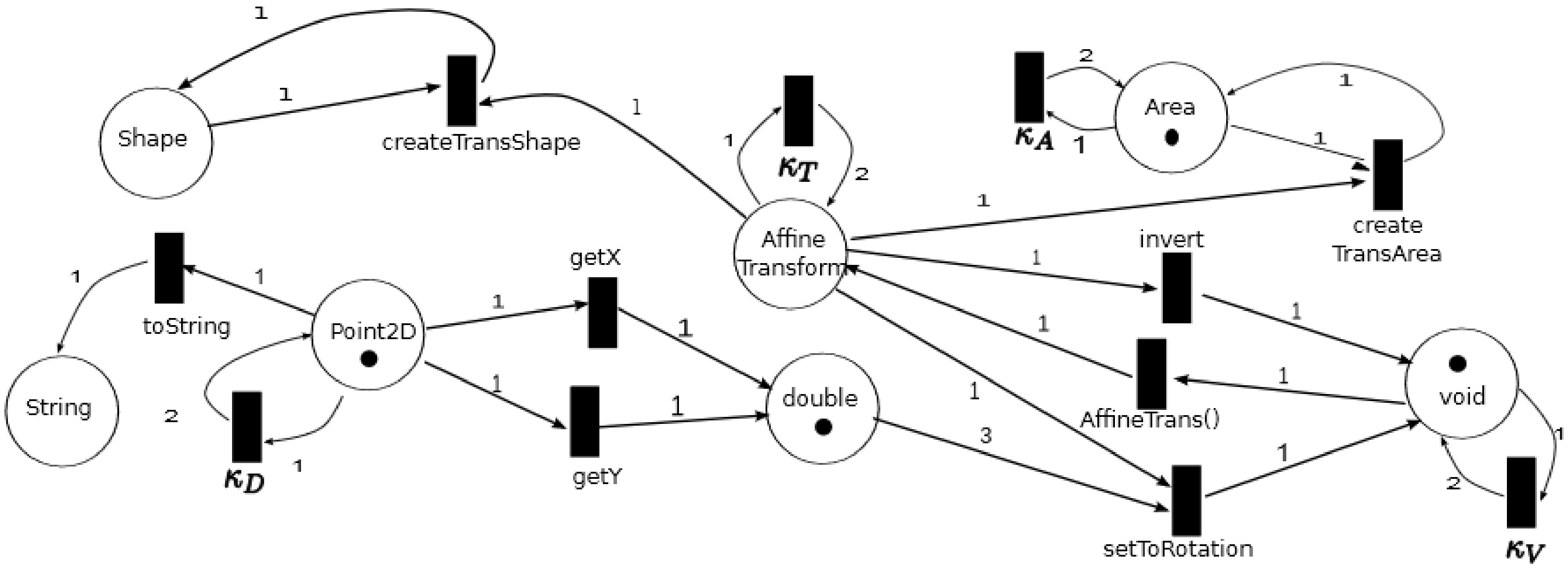
```
java.awt.geom
```

Output

```
Area rotate(Area obj, Point2D pt, double angle) {
  AffineTransform at = new AffineTransform();
  double x = pt.getX();
  double y = pt.getY();
  at.setToRotation(angle, x, y);
  Area obj2 = obj.createTransformedArea(at);
  return obj2;
}
```

Petri Nets

Library as a Petri Net



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

Discussion

Trade-off between work during and after building the representation:

- VSA/FTA: hard to build but easy to find a program
- TTN: easy to build but harder to find a program