Constraint Solving-Based Synthesis

Woosuk Lee

CSE9116 SPRING 2024

Hanyang University

Three Search Strategies

- **Enumerative**: enumeration + optimization
- **Stochastic**: probabilistic walk
- **Constraint-based**: encoding a synthesis problem as a SAT/SMT instance

Applications

- API synthesis (from ~1000 classes and 10000 methods available)

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;
}
```

https://utopia-group.github.io/sypet/

Applications

- SKETCH System

```
int W = 32;
void main(bit[W] x, bit[W] y){
        bit[W] xold = x;
        bit[W] yold = y;
        if(??){ x = x ^ y; }else{ y = x ^ y; }
        if(??){ x = x ^ y; }else{ y = x ^ y; }
        if(??){ x = x ^ y; }else{ y = x ^ y; }
        assert y == xold && x == yold;
}
```

```
int W = 32;
void main(bit[W] x, bit[W] y){
        bit[W] xold = x;
        bit[W] yold = y;
        y = x ^ y;
        x = x ^ y;
        y = x ^ y;
        assert y == xold && x == yold;
}
```

https://people.csail.mit.edu/asolar/

Applications

- Synthesizing sizable bit-twiddling tricks

P24(x) : Round up to the next highest power of 2

o_1:=bvsub (x,1)
o_2:=bvshr (o_1,1)
o_3:=bvor (o_1,o_2)
o_4:=bvshr (o_3,2)
o_5:=bvor (o_3,o_4)
o_6:=bvshr (o_5,4)
o_7:=bvor (o_5,o_6)
o_8:=bvshr (o_7,8)
o_9:=bvor (o_7,o_8)
o_{10}:=bvshr (o_9,16)
o_{11}:=bvor (o_9,o_{10})
res:=bvadd (o_{10},1)

P25(x, y) : Compute higher order half of product of x and y

o_1:=bvand (x,0xFFFF)
o_2:=bvshr (x,16)
o_3:=bvand (y,0xFFFF)
o_4:=bvshr (y,16)
o_5:=bvmul (o_1,o_3)
o_6:=bvmul (o_2,o_3)
o_7:=bvmul (o_1,o_4)
o_8:=bvmul (o_2,o_4)
o_9:=bvshr (o_5,16)
o_{10}:=bvadd (o_6,o_9)
o_{11}:=bvand (o_{10},0xFFFF)
o_{12}:=bvshr (o_{10},16)
o_{13}:=bvadd (o_7,o_{11})
o_{14}:=bvshr (o_{13},16)
o_{15}:=bvadd (o_{14},o_{12})
res:=bvadd (o_{15},o_8)

Key Idea

- Program = composition of components
- Step 1: **Encoding**: syntactic/semantic constraints → SAT/SMT formulas
- Step 2: Solving SAT/SMT
- Step 3: **Decoding**: Satisfying model → program

How to Encode?

- Brahma:
  - Oracle-guided Component-Based Program Synthesis, ICSE'10 (ACM/ IEEE 2020 Most Influential Paper Award)
  - https://github.com/fitzgen/synth-loop-free-prog
- SyPet:
  - Component-Based Synthesis for Complex APIs, POPL'17
  - https://github.com/utopia-group/sypet
- Sketch:
  - https://people.csail.mit.edu/asolar/

How to Encode?

- Brahma:
  - Oracle-guided Component-Based Program Synthesis, ICSE'10 (ACM/IEEE 2020 Most Influential Paper Award)
  - https://github.com/fitzgen/synth-loop-free-prog
- SyPet:
  - Component-Based Synthesis for Complex APIs, POPL'17
  - https://github.com/utopia-group/sypet
- Sketch:
  - https://people.csail.mit.edu/asolar/

Target Programs

- Straight-line code without loops
- viewed as a composition of usable components
  - Component: any function whose input-output relationship can be written as an SMT formula

Target Programs

Given: a **bag** of available components (=functions) [component_0 , … , component_{N-1}] (multiplicity matters)

```
synthesized_program(inputs...):
    temp0 ← component0(params0...)
    temp1 ← component1(params1...)
    // ...
    tempN−1 ← componentN−1(paramsN−1...)
    return tempN−1
```

Target Programs

- With parameter variable x and the following components
  - function f whose arity is 1
  - function g whose arity is 2
- Examples that **can be synthesized**

```
tmp0 ← f(x)
tmp1 ← g(tmp0, x)
return tmp1
```

```
tmp0 ← g(x, x)
tmp1 ← f(tmp0)
return tmp1
```

```
tmp0 ← f(x)
tmp1 ← g(x, x)
return tmp1
```

May contain redundant lines

Target Programs

- With parameter variable x and the following components
  - function f whose arity is 1
  - function g whose arity is 2
- Examples that **cannot be synthesized**

```
tmp0 ← f(x)
tmp1 ← f(x)
tmp2 ← g(tmp0, tmp0)
return tmp2
```

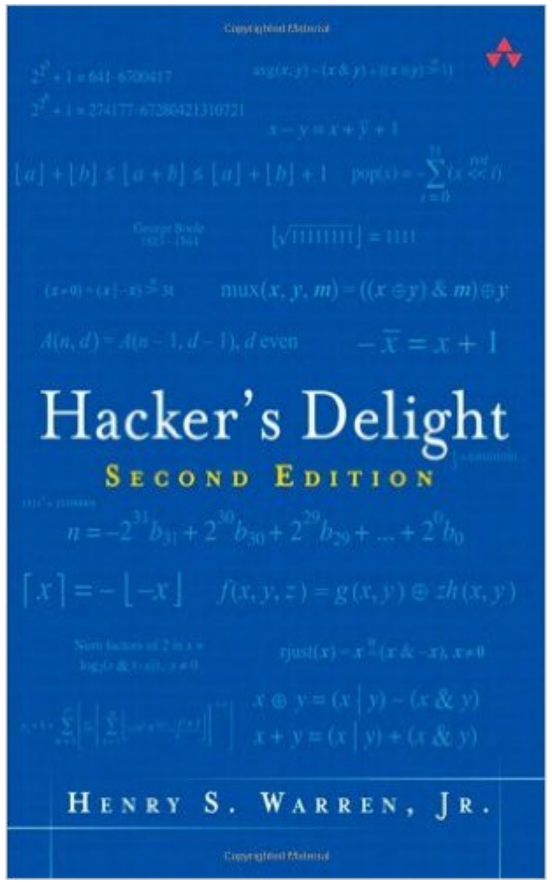
f is allowed only once!

```
tmp0 ← g(x, x)
tmp1 ← h(x)
return tmp1
```

h is not allowed!

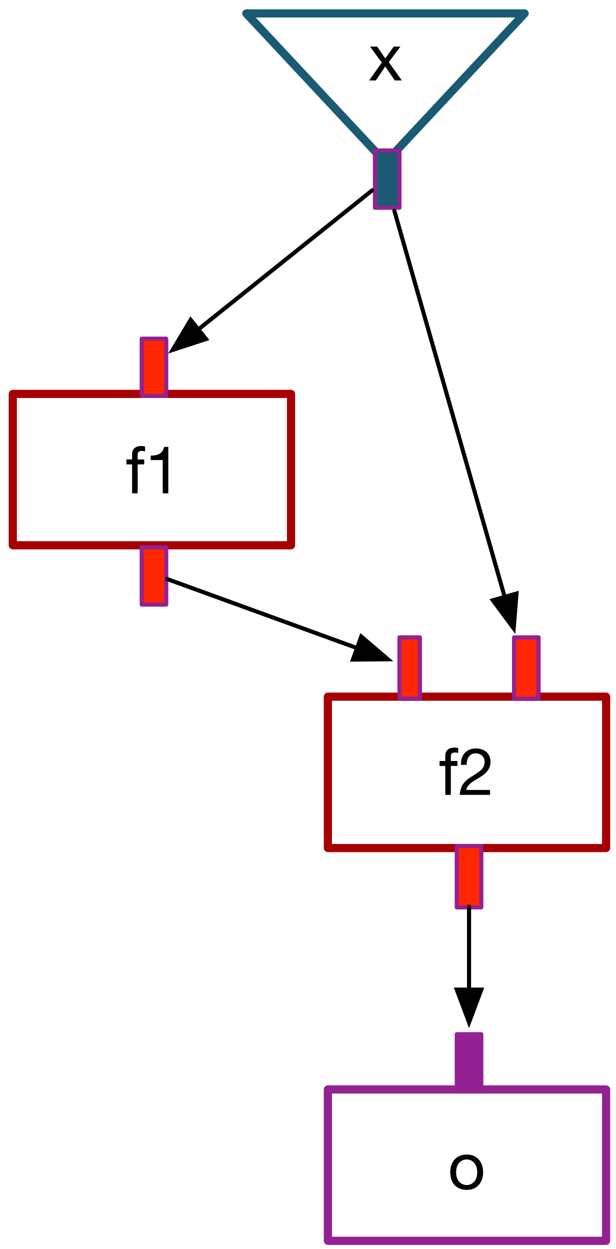
Example: Hackers Delight

- Change rightmost contiguous 1's to 0's
- Target: `f(BitVec x) : BitVec`
- Components :
  - `f1(a) = a - 1`
  - `f2(a, b) = a & b`



- Constraints: `f(01100) = 01000, f(10001) = 10000, …`
- Solution: `f(x) = x & (x - 1)`

Program as DAG

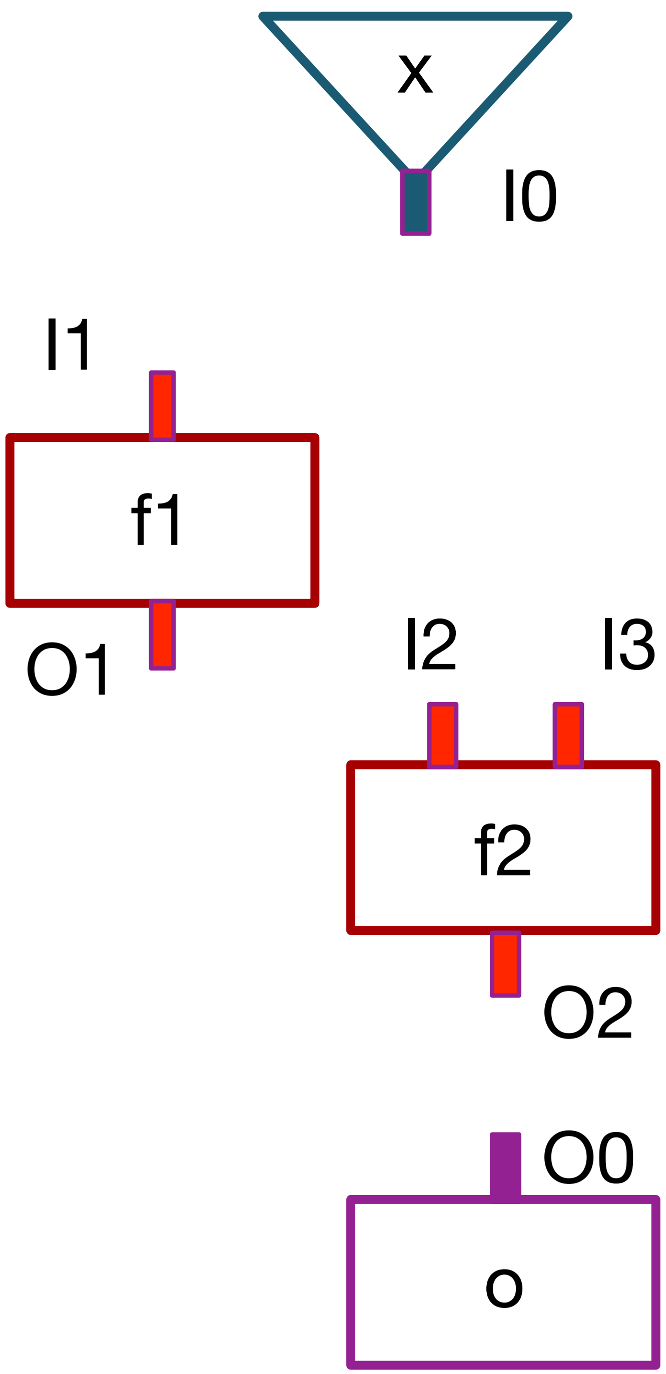



```
Components:
f1(a) = a - 1
f2(a, b) = a & b

Solution:
1: O1 = f1(x)
2: O2 = f2(x, O1)
```

↑ **Line number**

IDs of Inputs / Outputs of Components



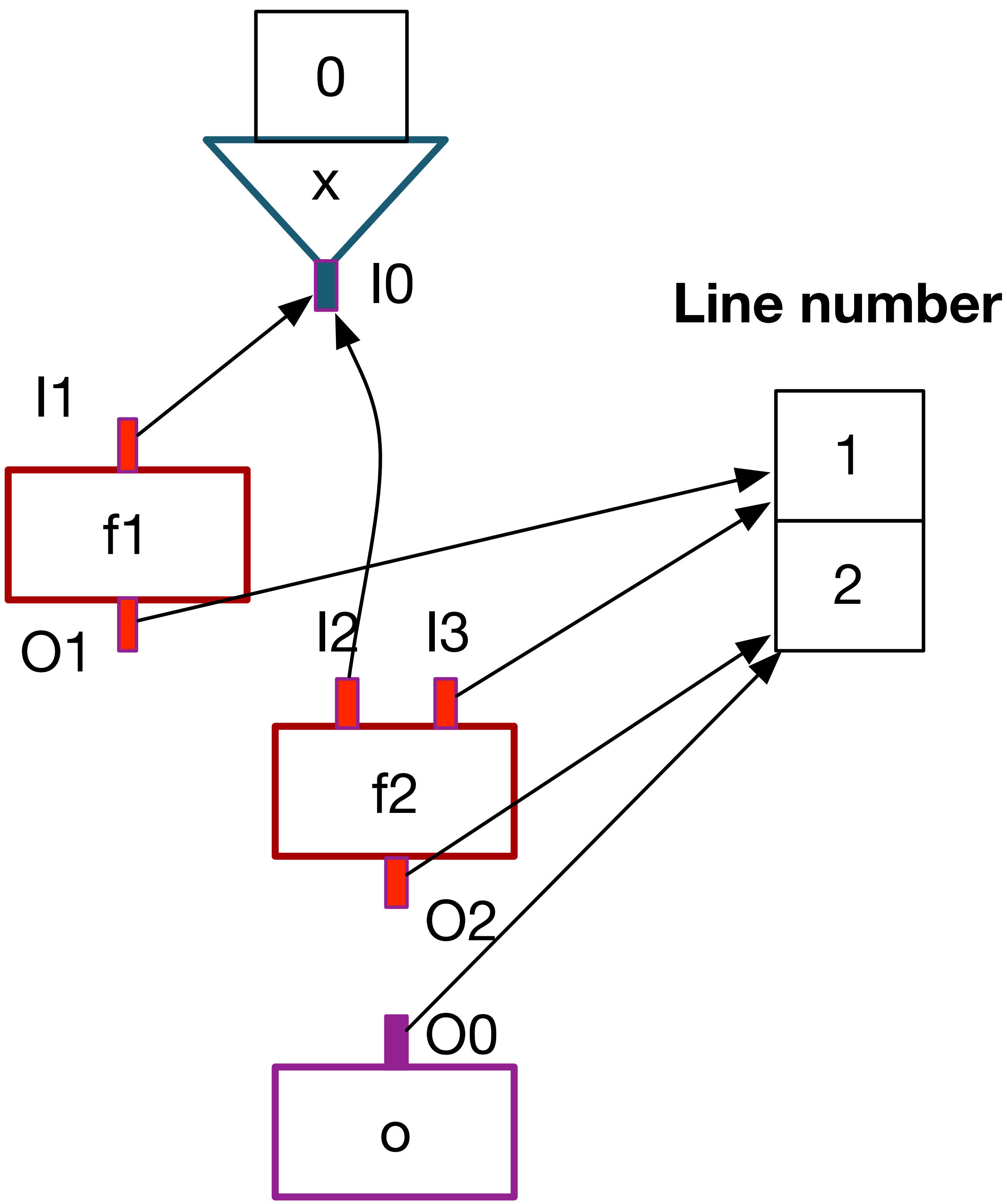
Components:

```
f1(a) = a - 1
f2(a, b) = a & b
```

Solution:

```
1: O1 = f1(x)
2: O2 = f2(x, O1)
```

Connecting Components



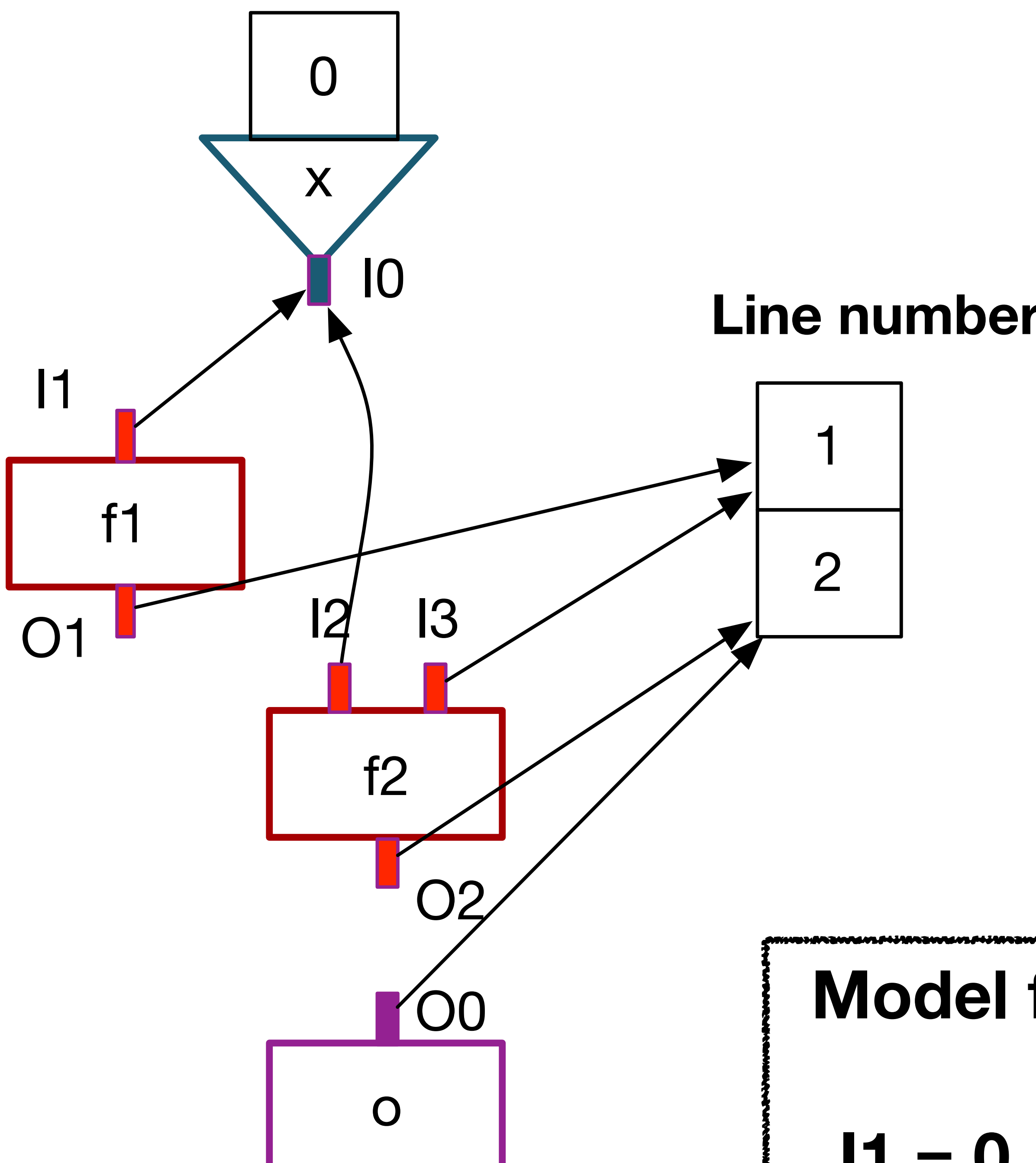
Components:

```
f1(a) = a - 1
f2(a, b) = a & b
```

Solution:

```
1: O1 = f1(x)
2: O2 = f2(x, O1)
```

Connecting Components



Components:

```
f1(a) = a - 1
f2(a, b) = a & b
```

Solution:

```
1: O1 = f1(x)
2: O2 = f2(x, O1)
```

Model for the solution:

I1 = 0 I2 = 0 I3 = 1
O1 = 1 O2 = 2 O0 = 2

SMT Encoding

- Parameter vars. of components

$$\mathbf{P} := \{I_1, I_2, I_3\}$$

- Output vars. of components

$$\mathbf{R} := \{O_1, O_2\}$$

- Location vars. for connecting components

$$L := \{l_x \mid x \in \mathbf{P} \cup \mathbf{R}\}$$

Syntactic Constraint

Possible range of parameter vars

Possible range of output var

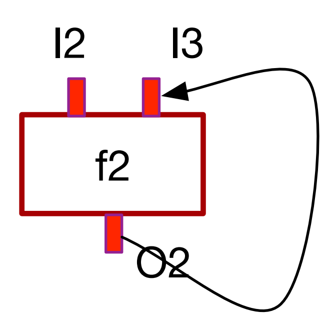
$$\psi_{\texttt{wfp}} := \bigwedge_{x \in \mathbf{P}} (0 \leq l_x < 3) \wedge \bigwedge_{x \in \mathbf{R}} (1 \leq l_x < 3) \wedge \psi_{\texttt{cons}}(L) \wedge \psi_{\texttt{acyc}}(L)$$

$$\psi_{\texttt{cons}} := \bigwedge_{x,y \in \mathbf{R}, x \neq y} l_x \neq l_y$$

One component at a line

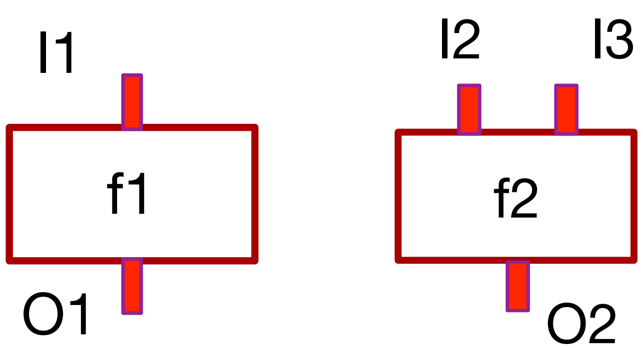
$$\psi_{\texttt{acyc}} := l_{I_1} < l_{O_1} \wedge l_{I_2} < l_{O_2} \wedge l_{I_3} < l_{O_2}$$

Must use already defined ones:



← prohibited!

Library Specification

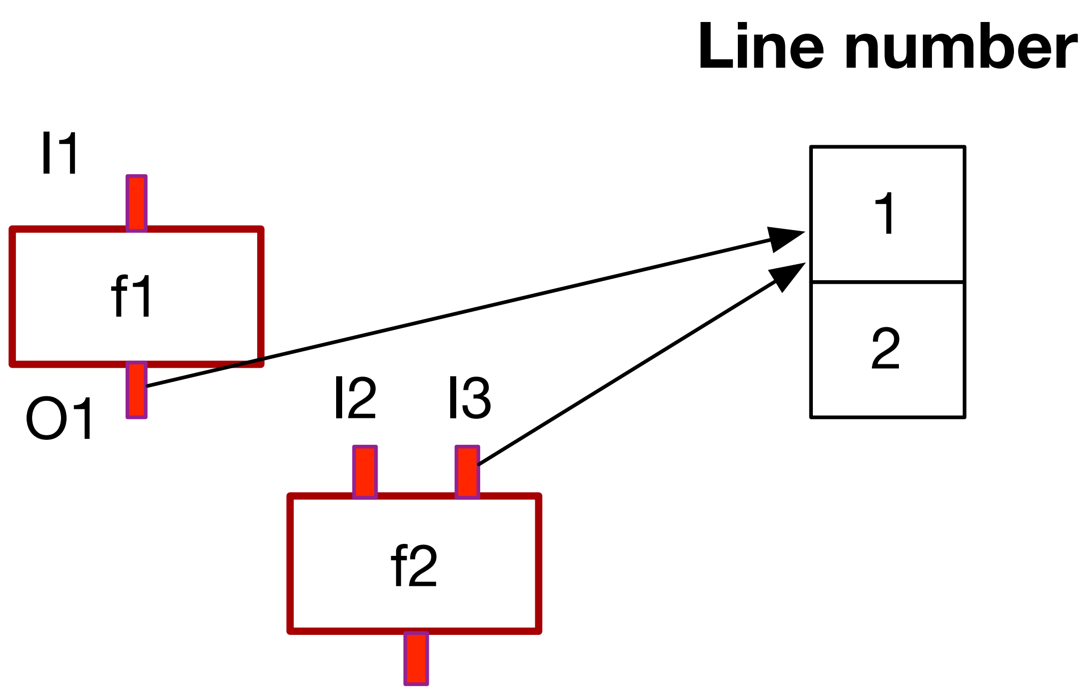


Components:

```
f1(a) = a - 1
f2(a, b) = a & b
```

$$\phi_{\texttt{lib}} = [O_1 = I_1 - 1] \wedge [O_2 = I_2 \ \& \ I_3]$$

Connecting Components



O1 = I3 !

$$\phi_{\texttt{conn}} = \bigwedge_{x,y \in \mathbf{P} \cup \mathbf{R} \cup \{I_0, O_0\}} (l_x = l_y \implies x = y)$$

Final SMT Formula

- For brevity, assume a single I/O example

Usable components

I/O constraint

$$\psi_{\texttt{wfp}} \wedge \phi_{\texttt{lib}} \wedge \phi_{\texttt{conn}} \wedge (I_0 = 01100) \wedge (O_0 = 01000)$$

Programs should look good

Encoding of data flows

Properties

- Decisive performance factor: size of library
- Relying on modern SMT solvers with performance being continuously improved
- Multiplicity constraints
  - Must use some operator ≤ n times ← Hard to specify using a CFG

Application of Brahma: Program Repair

```
int is_upward_preferred(int inhibit, int up_sep,
          int down_sep) {
  int bias;
  if(inhibit)
    bias = down_sep; //fix: bias=up_sep+100
  else
    bias = up_sep;
  if (bias > down_sep)
    return 1;
  else
    return 0;
}
```

Fig. 1. Code excerpt from `Tcas`

S. Mechtaev, J. Yi, A. Roychoudhury, Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis, ICSE'16

Passed / Failed Test Cases

TABLE I
A TEST SUITE FOR THE PROGRAM IN FIG. 1

Test	Inputs			Expected output	Observed output	Status
	inhibit	up_sep	down_sep			
1	1	0	100	0	0	pass
2	1	11	110	1	0	fail
3	0	100	50	1	1	pass
4	1	-20	60	1	0	fail
5	0	0	10	0	0	pass

Statistical Fault Localization

TABLE II
TARANTULA FAULT LOCALIZATION RESULT ON THE PROGRAM IN FIG. 1

Line	Score	Rank
4	0.75	1
10	0.6	2
3	0.5	3
7	0.5	3
6	0	5
8	0	5

Suspicious score for each statement s:

$$susp(s) = \frac{failed(s)/totalfailed}{passed(s)/totalpassed + failed(s)/totalfailed}$$

Patch Constraint Generation via Symbolic Execution

Test	Inputs			Expected output	Observed output	Status
	inhibit	up_sep	down_sep			
1	1	0	100	0	0	pass
2	1	11	110	1	0	fail
3	0	100	50	1	1	pass
4	1	-20	60	1	0	fail
5	0	0	10	0	0	pass

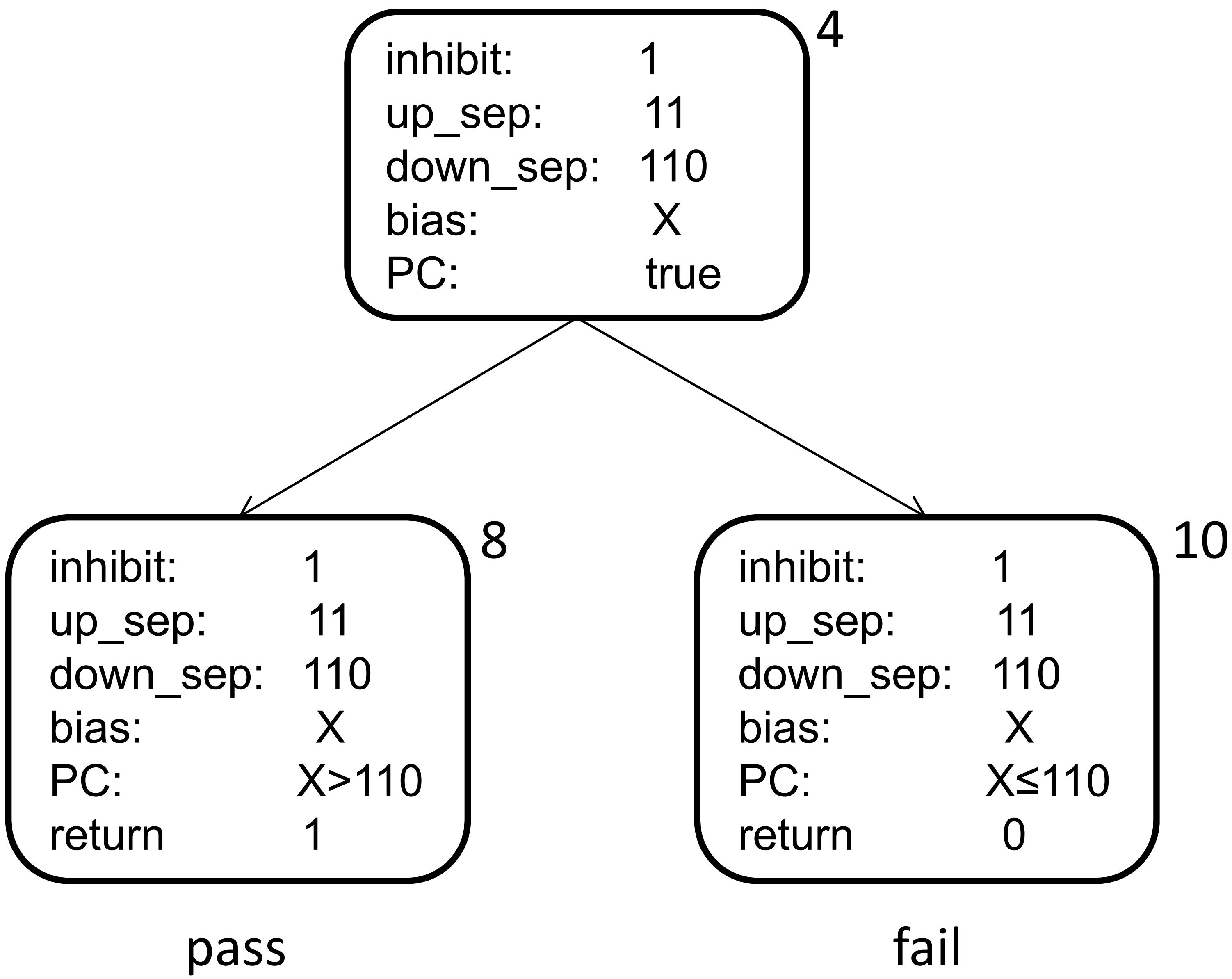
```
int is_upward_preferred(int inhibit, int up_sep,
      int down_sep) {
  int bias;
  if(inhibit)
    bias = f(inhibit, up_sep, down_sep)
  else
    bias = up_sep;
  if (bias > down_sep)
    return 1;
  else
    return 0;
}
```

Concrete execution

Symbolic execution with `bias` replaced with a symbolic variable

Fig. 1. Code excerpt from Tcas

Two Possible Execution Flows



Patch Constraint Generation via Symbolic Execution

Test	Valuations: inhibit	Valuations: up_sep	Valuations: down_sep	Constraint over f
1	1	0	100	bias ≤ down_sep ⇔ f(1,0,100) ≤ 100
2	1	11	110	bias > down_sep ⇔ f(1,11,110) > 110
4	1	-20	60	bias > down_sep ⇔ f(1,-20,60) > 60

Patch Generation as Synthesis

- Target: f (inhibit: int, up_sep: int, down_sep: int) : int

- Syntactic constraint

$$
\begin{array}{rcl}
S & \rightarrow & \text{inhibit} \mid \text{up\_sep} \mid \text{down\_sep} \mid 0 \mid 1 \mid \cdots \\
 & \mid & S + S \mid S - S \mid S \times S \mid S/S
\end{array}
$$

- Semantic constraint

$$f(1, 11, 110) > 110 \wedge f(1, 0, 100) \leq 100 \wedge f(1, -20, 60) > 60$$

- Solving with component-based synthesis:

```
f(inhibit, up_sep, down_sep) = up_sep + 100
```

How to Encode?

- Brahma:
  - Oracle-guided Component-Based Program Synthesis, ICSE'10 (ACM/IEEE 2020 Most Influential Paper Award)
  - https://github.com/fitzgen/synth-loop-free-prog
- SyPet:
  - Component-Based Synthesis for Complex APIs, POPL'17
  - https://github.com/utopia-group/sypet
- Sketch:
  - https://people.csail.mit.edu/asolar/

API Synthesis

- Input: (1) Usable API functions,
  (2) Problem: Signature of target function + unit test cases
- Output: straight line code that consists of API functions

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

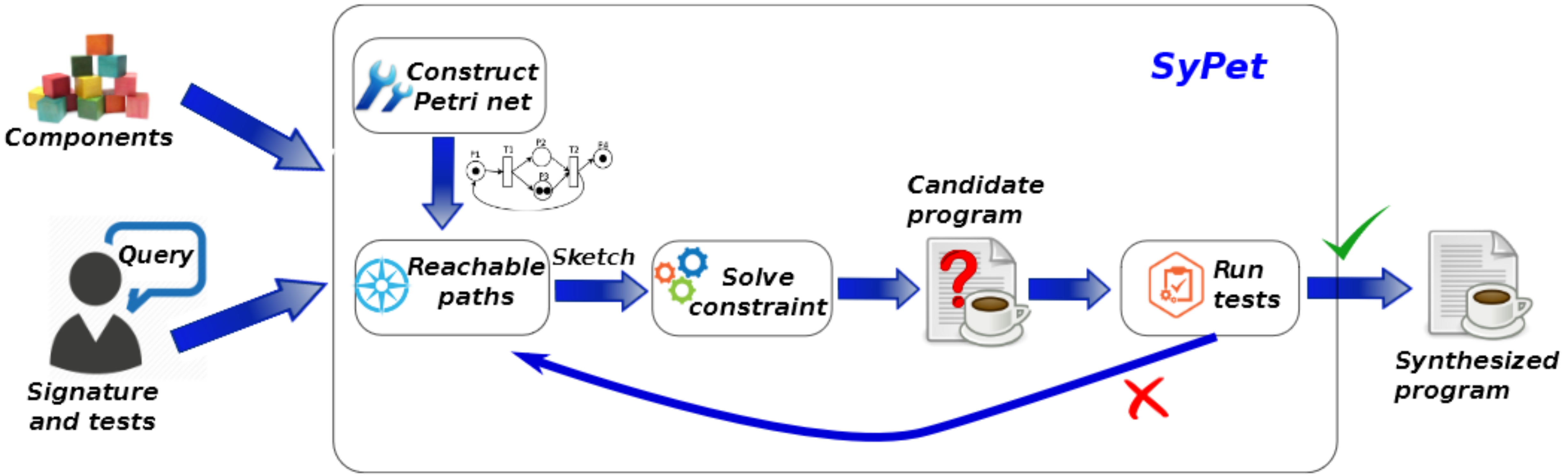
```
Area rotate(Area obj, Point2D pt, double angle) {
  AffineTransform at = new AffineTransform();
  double x = pt.getX();
  double y = pt.getY();
  at.setToRotation(angle, x, y);
  Area obj2 = obj.createTransformedArea(at);
  return obj2;
}
```

Too many usable API functions
Naive enumeration won't work!

Key Idea

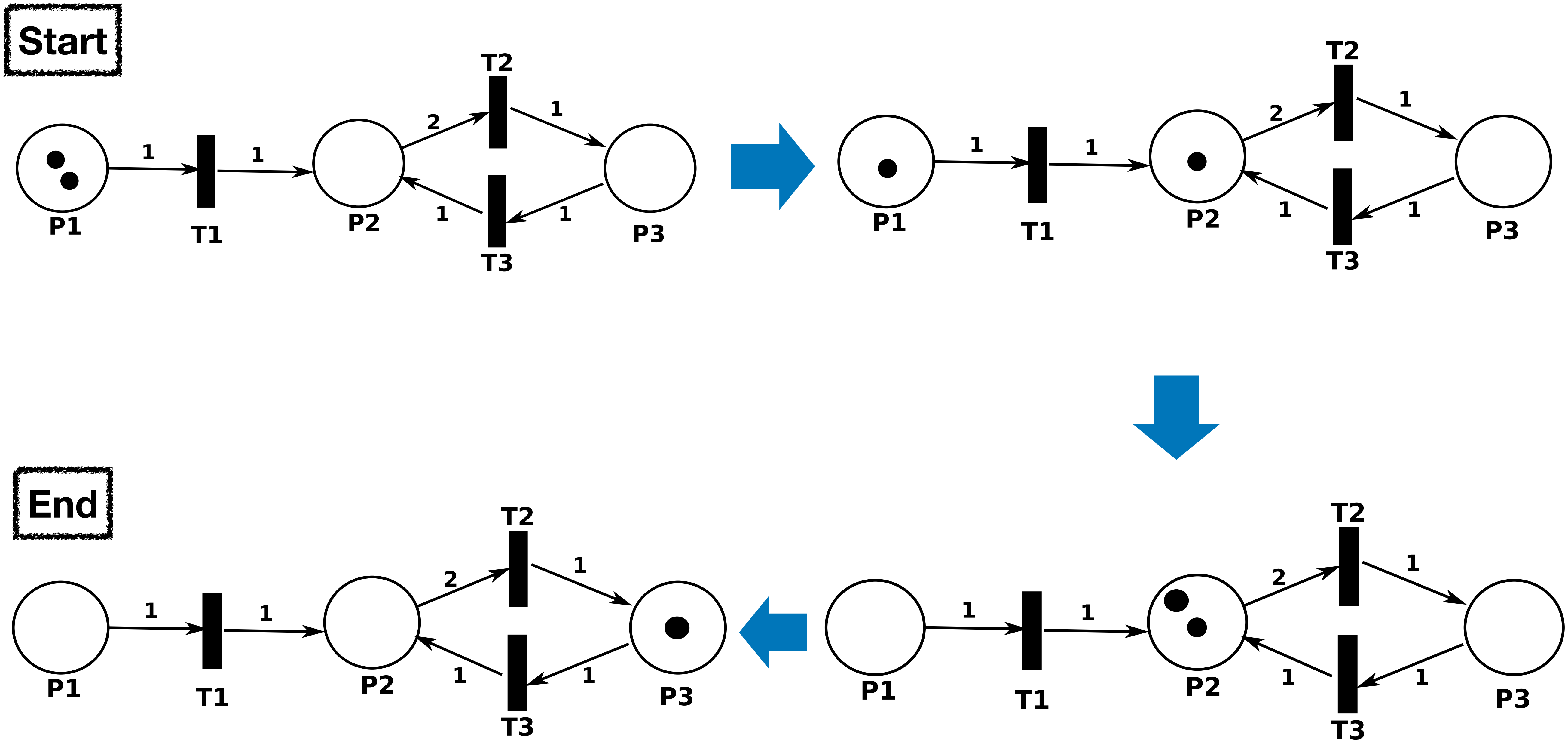
- Step 1: Construct a graph
  - Node: Type
  - Edge: single invocation of API function
- Step 2: Find a path from parameter types to return type
  - Using SAT or ILP (integer linear programming)
- Step 3: Decode the path into a program

SyPet



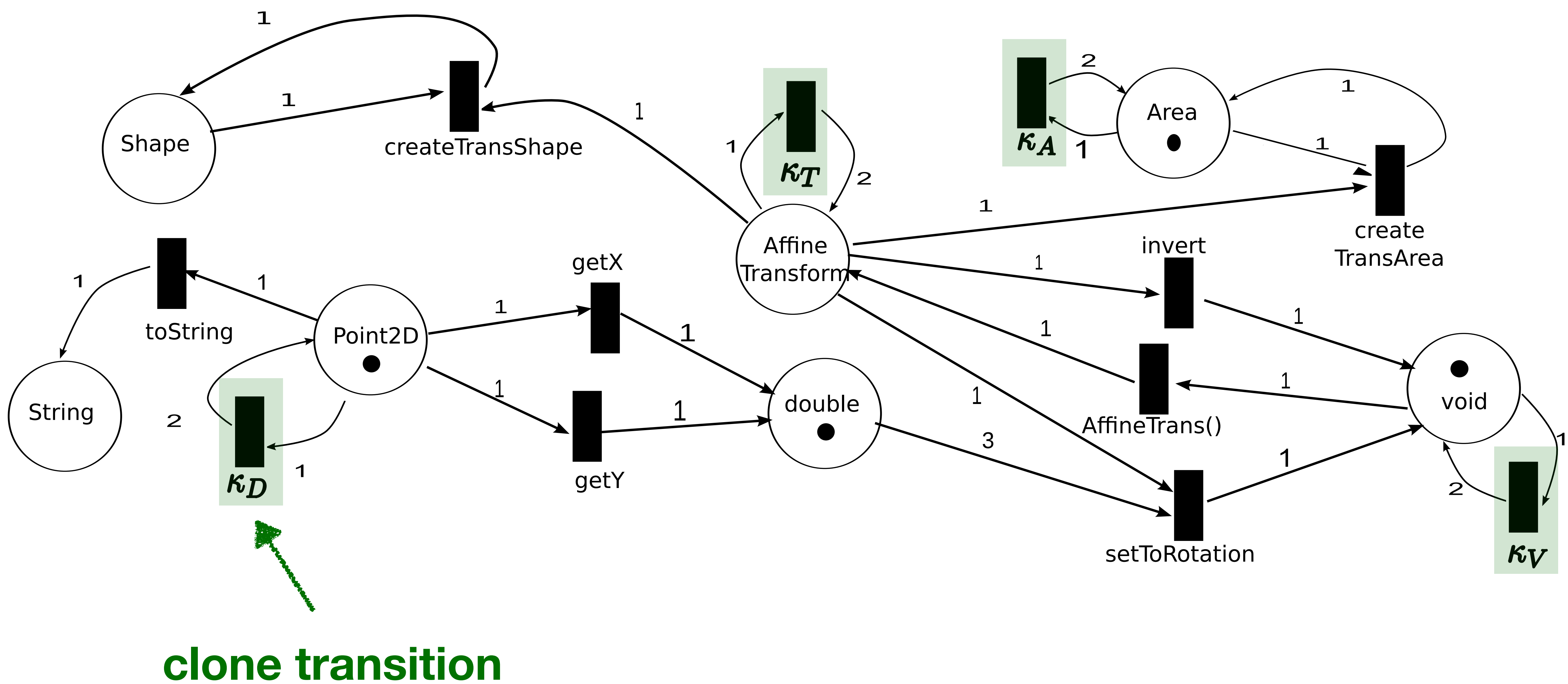
Components
Query
Signature
and tests
SyPet
Construct
Petri net
Reachable
paths
Sketch
Solve
constraint
Candidate
program
Run
tests
Synthesized
program

Petri Nets



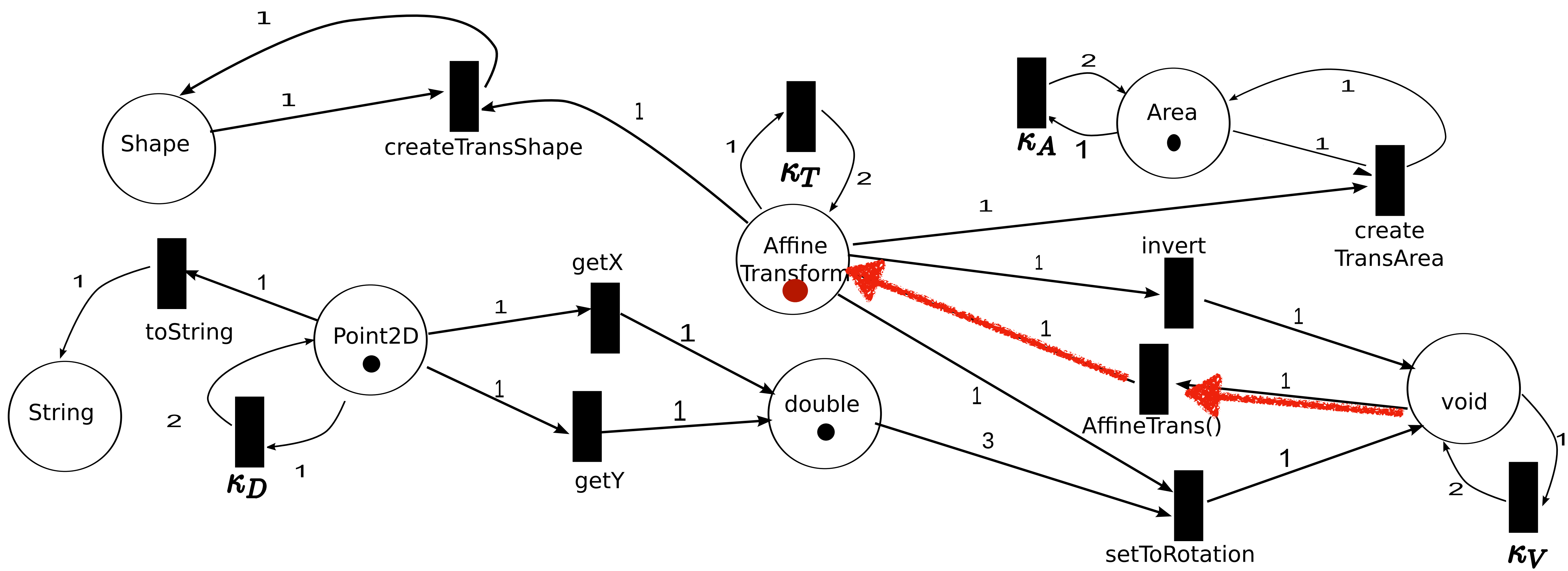
Petri Net Path = Well-Typed Program

```
Area rotate(Area obj, Point2D pt, double angle)
```



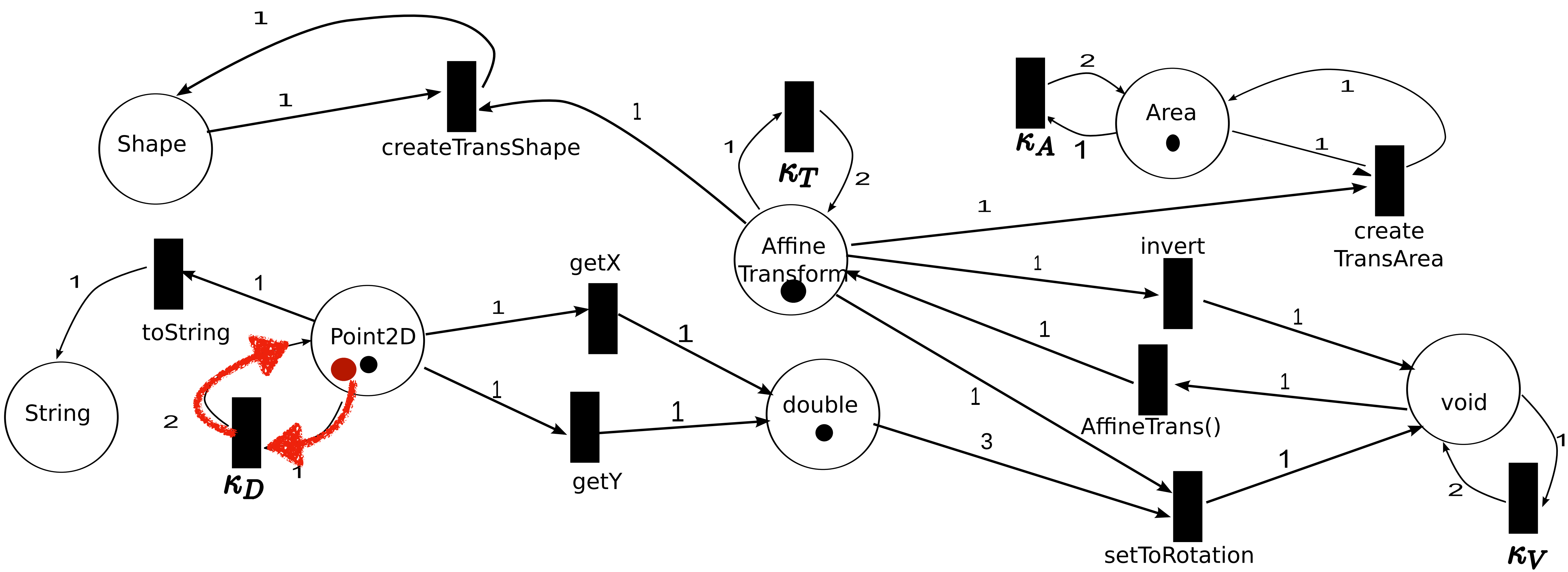
Petri Net Path = Well-Typed Program

```
Area rotate(Area obj, Point2D pt, double angle)
```



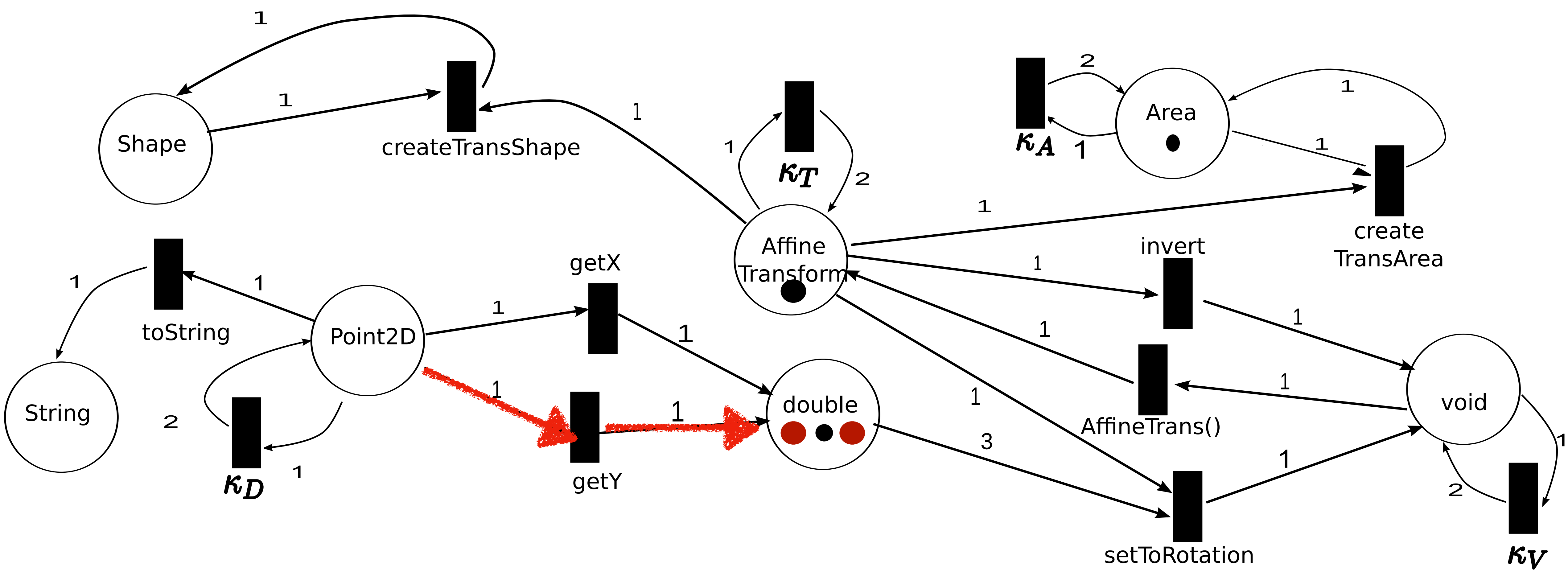
Petri Net Path = Well-Typed Program

```
Area rotate(Area obj, Point2D pt, double angle)
```



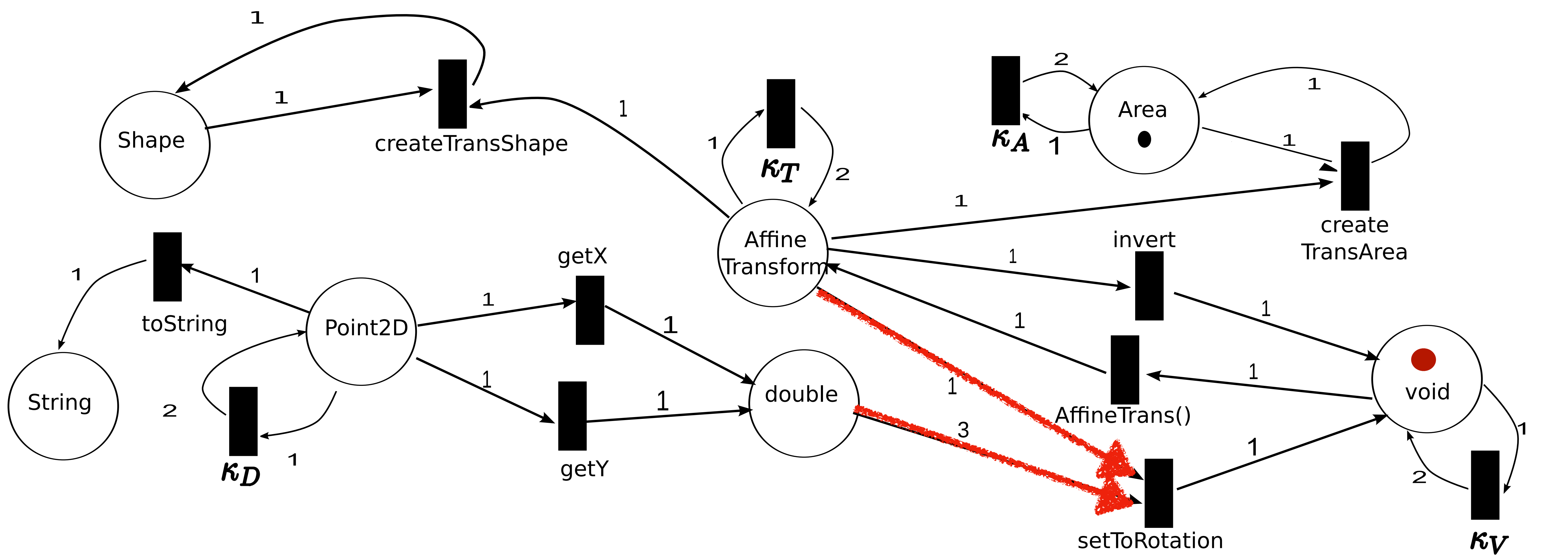
Petri Net Path = Well-Typed Program

`Area rotate(Area obj, Point2D pt, double angle)`



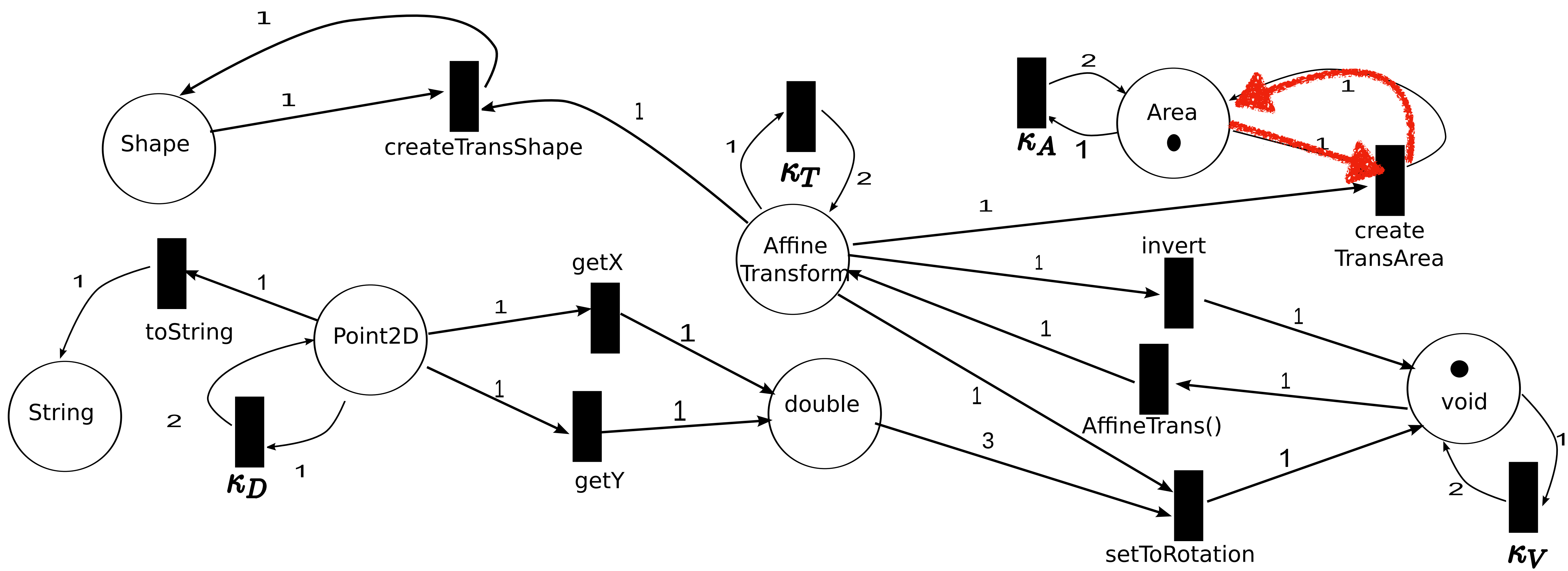
Petri Net Path = Well-Typed Program

```
Area rotate(Area obj, Point2D pt, double angle)
```



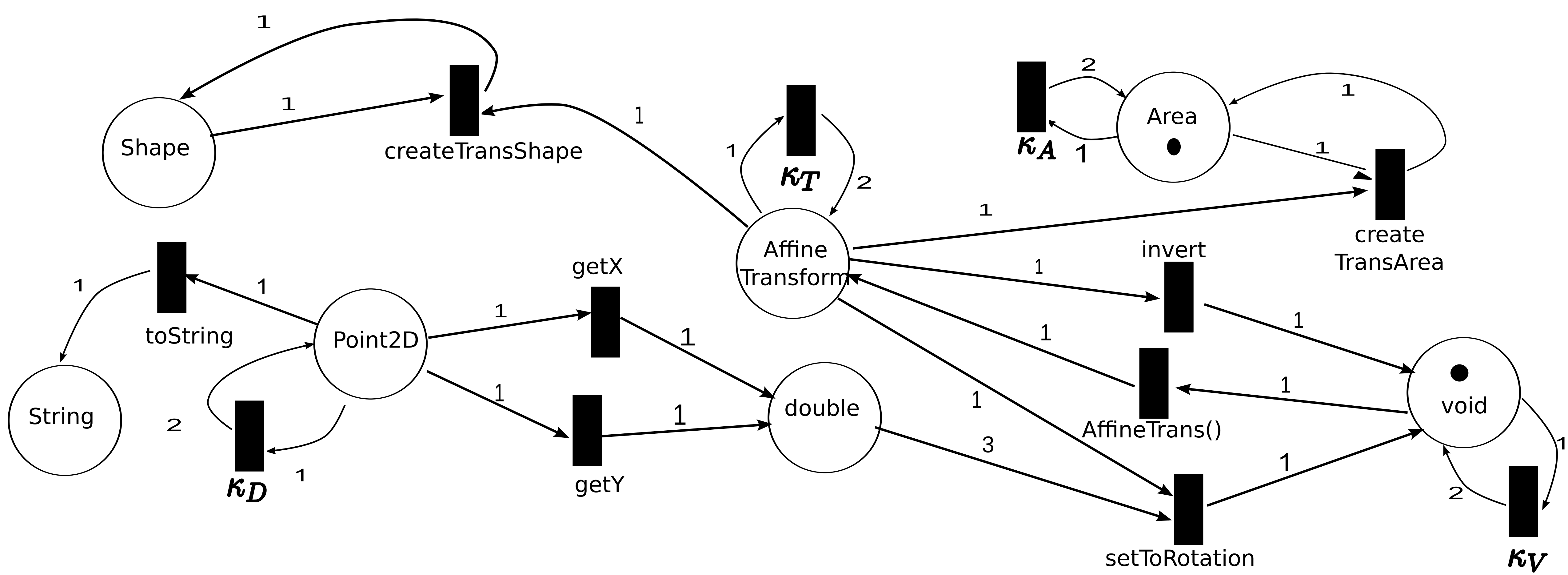
Petri Net Path = Well-Typed Program

```
Area rotate(Area obj, Point2D pt, double angle)
```



Petri Net Path = Well-Typed Program

```
Area rotate(Area obj, Point2D pt, double angle)
```



Path found!

Final state: a token for the return type

(+ tokens for the void type (for supporting side-effects))

Petri Net Path = Well-Typed Incomplete Program

- Generate the following sketch from the path

```
x = #1.getX(); y = #2.getY();
t = new AffineTransform();
#3.setToRotation(#4, #5, #6);
a = #7.createTransformedArea(#8);
return #9;
```

- Try to fill #1 ~ #9 with all possible variables to find a correct program wrt test cases

- Search for another petri net path if no program can be found.

Properties

- Pros: scalable wrt #. of API functions supporting side effects
  - See: *Program synthesis by type-guided abstraction refinement*, POPL'20 for an SMT encoding of petri-net reachability
- Cons: cannot support conditionals and loops
  - See: *FrAngel: Component-Based Synthesis with Control Structures*, POPL'19 for how to support conditionals and loops https://github.com/kensens/FrAngel
- Affects Hoogle for Haskell API search
  - https://hoogleplus.goto.ucsd.edu

How to Encode?

- Brahma:
  - Oracle-guided Component-Based Program Synthesis, ICSE'10 (ACM/IEEE 2020 Most Influential Paper Award)
  - https://github.com/fitzgen/synth-loop-free-prog
- SyPet:
  - Component-Based Synthesis for Complex APIs, POPL'17
  - https://github.com/utopia-group/sypet
- Sketch:
  - https://people.csail.mit.edu/asolar/

Example: Swap w/o a Temp Variable

```
generator int sign() {
  if ?? {return 1;} else {return -1;}
}

void swap (int& x, int& y) {
  x = x + sign() * y;
  y = x + sign() * y;
  x = x + sign() * y;
}

harness void main (int x, int y) {
  int tx = x;
  int ty = y;
  swap (x, y);
  assert (x == ty && y == tx);
}
```

Example: Swap w/o a Temp Variable

```
generator int sign() {
  if ?? {return 1;} else {return -1;}
}

void swap (int& x, int& y) {
  x = x + sign() * y;
  y = x + sign() * y;
  x = x + sign() * y;
}
```

```
{x ↦ X, y ↦ Y}
```

```
{x ↦ X + (ite (??_1) 1 -1) * Y, y ↦ Y}
```

```
{x ↦ X + (ite (??_1) 1 -1) * Y,
 y ↦ X + (ite (??_1) 1 -1) * Y +
     (ite (??_2) 1 -1) * Y}
```

```
{x ↦ X + (ite (??_1) 1 -1) * Y +
     (ite (??_3) 1 -1) *
      (X + (ite (??_1) 1 -1) * Y +
      (ite (??_2) 1 -1) * Y),
 y ↦ X + (ite (??_1) 1 -1) * Y +
     (ite (??_2) 1 -1) * Y}
```

Example: Swap w/o a Temp Variable

```
…
harness void main (int x, int y) {
  int tx = x;
  int ty = y;
  swap (x, y);
  assert (x == ty && y == tx);
}
```

```
{x ↦ X + (ite (??1) 1 -1) * Y +
     (ite (??3) 1 -1) *
      (X + (ite (??1) 1 -1) * Y +
      (ite (??2) 1 -1) * Y),
 y ↦ X + (ite (??1) 1 -1) * Y +
     (ite (??2) 1 -1) * Y}
```

Find holes such that

Where does it come from? Through CEGIS

```
∀ X, Y ∈ CEXs.
  X + (ite (??1) 1 -1) * Y + (ite (??3) 1 -1)
    * (X + (ite (??1) 1 -1) * Y + (ite (??2) 1 -1) * Y) = Y
  ∧ X + (ite (??1) 1 -1) * Y + (ite (??2) 1 -1) * Y = X
```

➡ $??_1$ ↦ true, $??_2$ ↦ false, $??_3$ ↦ false

Other Details

- RegExp for specify usable operators and operands can be used to fill holes
- What about loops and recursive functions?
  - They are unrolled finite times (adjustable via options)
- To handle non-linear integer arithmetic beyond the capability of SMT
  - integers are bounded
  - integer operations are encoded as lookup tables
  - and then a SAT solver is used.

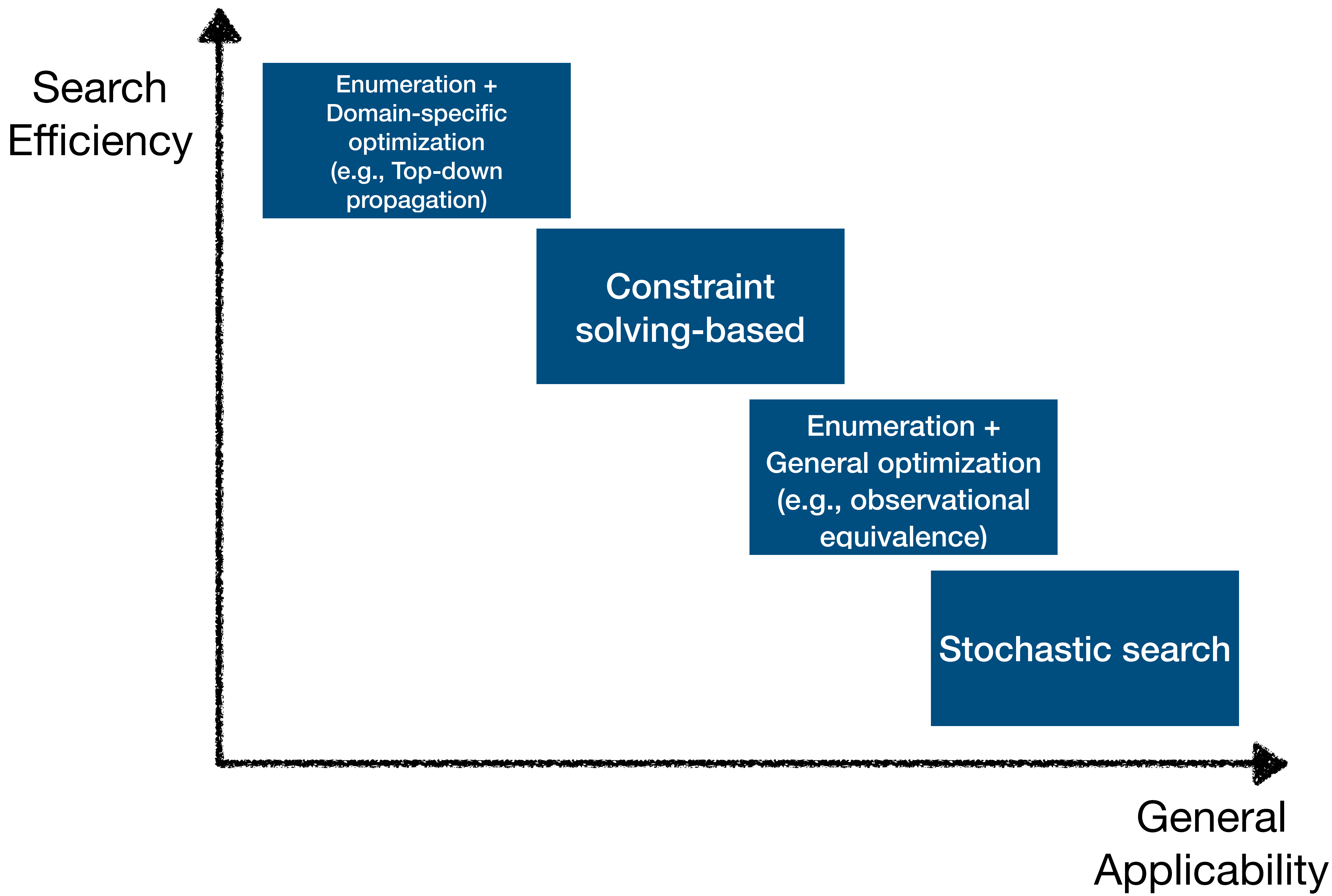
Limitations of Sketch

- Loops, integers are bounded.
- Not easy to specify Sketch
  - But as search gets better, user input can be simplified
- Cannot guide the search towards more likely programs

Summary

- Encoding: synthesis constraints → SAT/SMT formulas, Decoding: model → solution
- Can express syntactic constraints beyond the power of CFGs
- Overall performance heavily relies on the performance of SAT/SMT solvers.

Efficiency vs. Applicability



Efficiency vs. Applicability