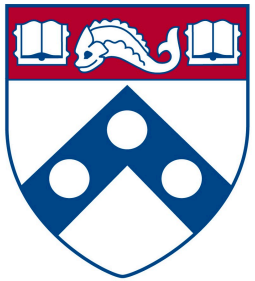
Deep Reinforcement Learning for Program Verification and Synthesis

Xujie Si, Hanjun Dai, Yuan Yang, Mukund Raghothaman, Mayur Naik, Le Song



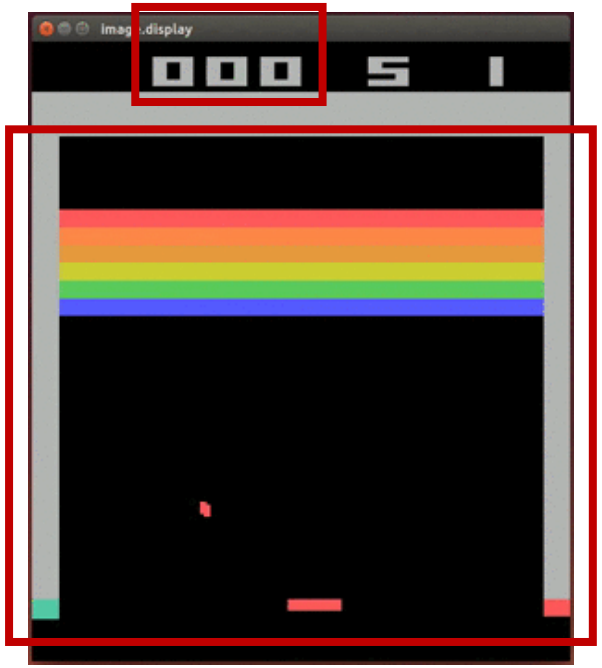
University of Pennsylvania
Georgia Institute of Technology

The Success of Deep Reinforcement Learning

Reward

State

"fix-sized 2D raw pixels"



Action

(move left, move right)

Neural Policy

$P(action \mid state)$

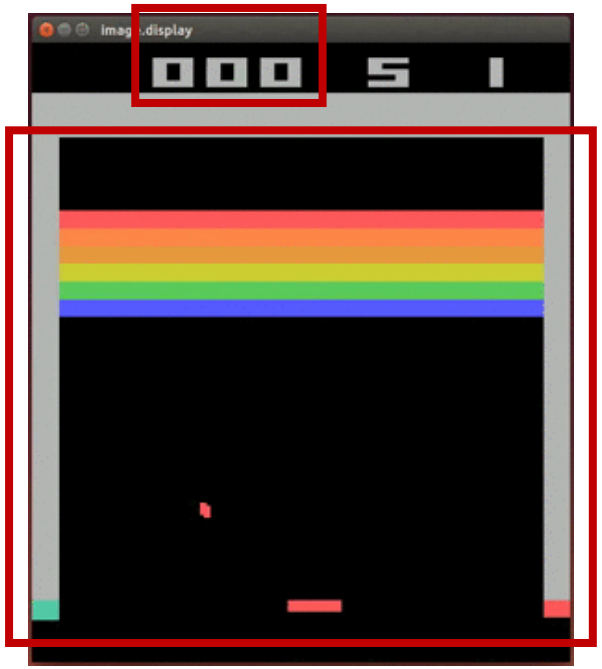
"automatically learned from self-play"

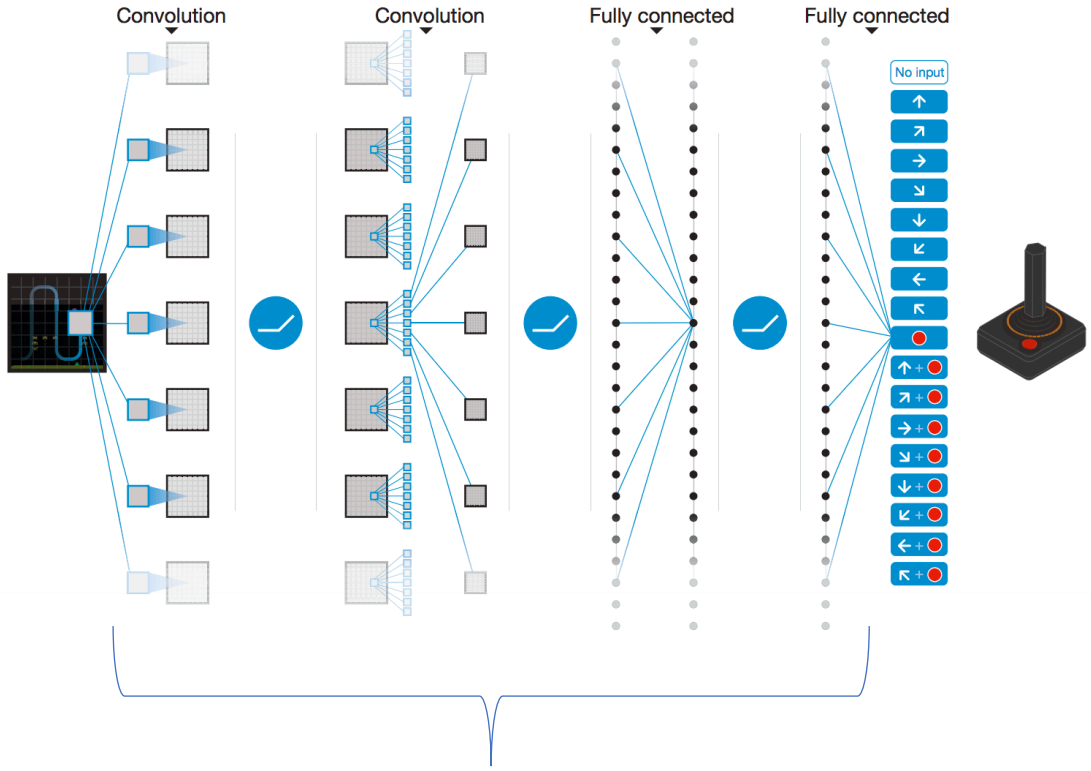
The Success of Deep Reinforcement Learning

Reward

State

"fix-sized 2D raw pixels"





Action

(move left, move right)

Neural Policy

$$\mathrm{P}(action \mid state)$$

"automatically learned from self-play"

DRL for program reasoning

Atari Games	raw pixels size is fixed	Convolutional Neural Network	Simple & Limited number of actions
Program			

DRL for program reasoning

Atari Games	raw pixels size is fixed	Convolutional Neural Network	Simple & Limited number of actions
Program	***structured*** data size ***varies***	***Graph Neural Network***	***Actions specified in a Context-free Grammar***

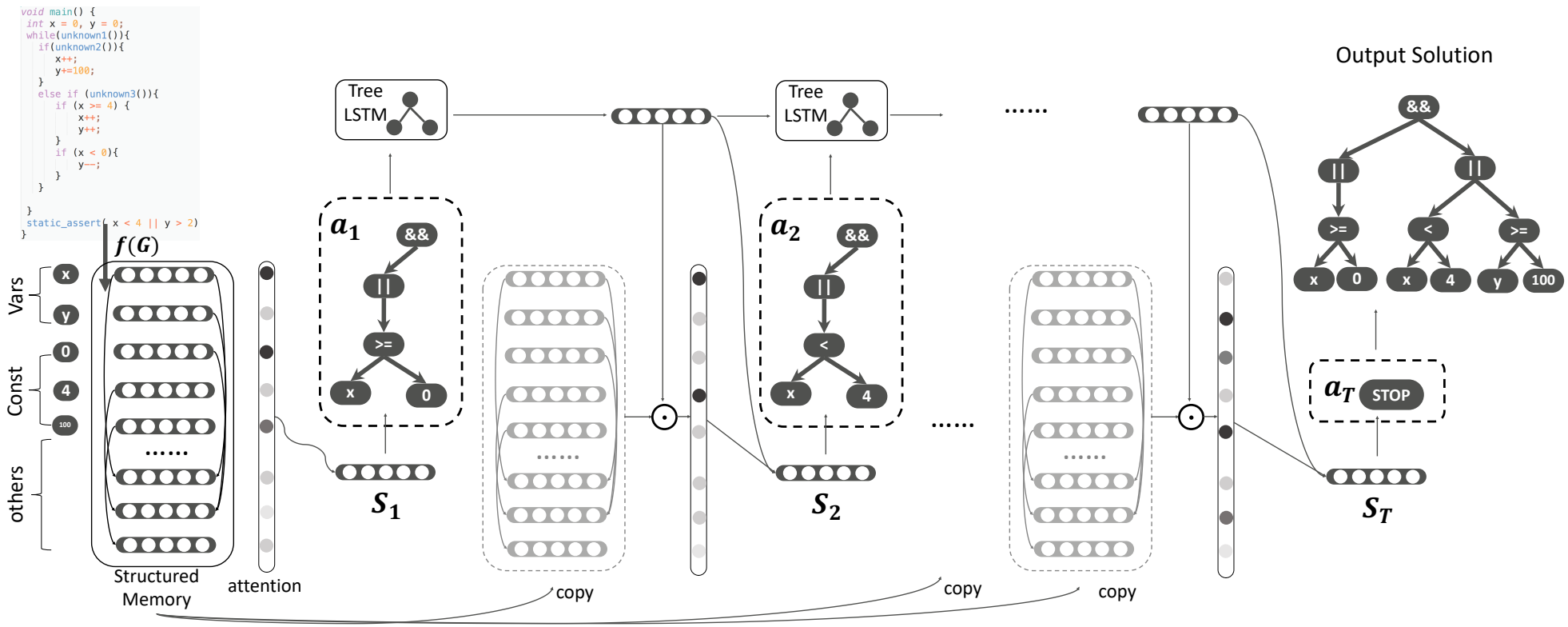
Learning loop invariant

Program	Loop Invariant
$x := -50;$ **while** $(x<0)$ { $\quad x := x+y;$ $\quad y := y+1$ } **assert**$(y>0)$	$(x<0 \vee y>0)$

Requirement:

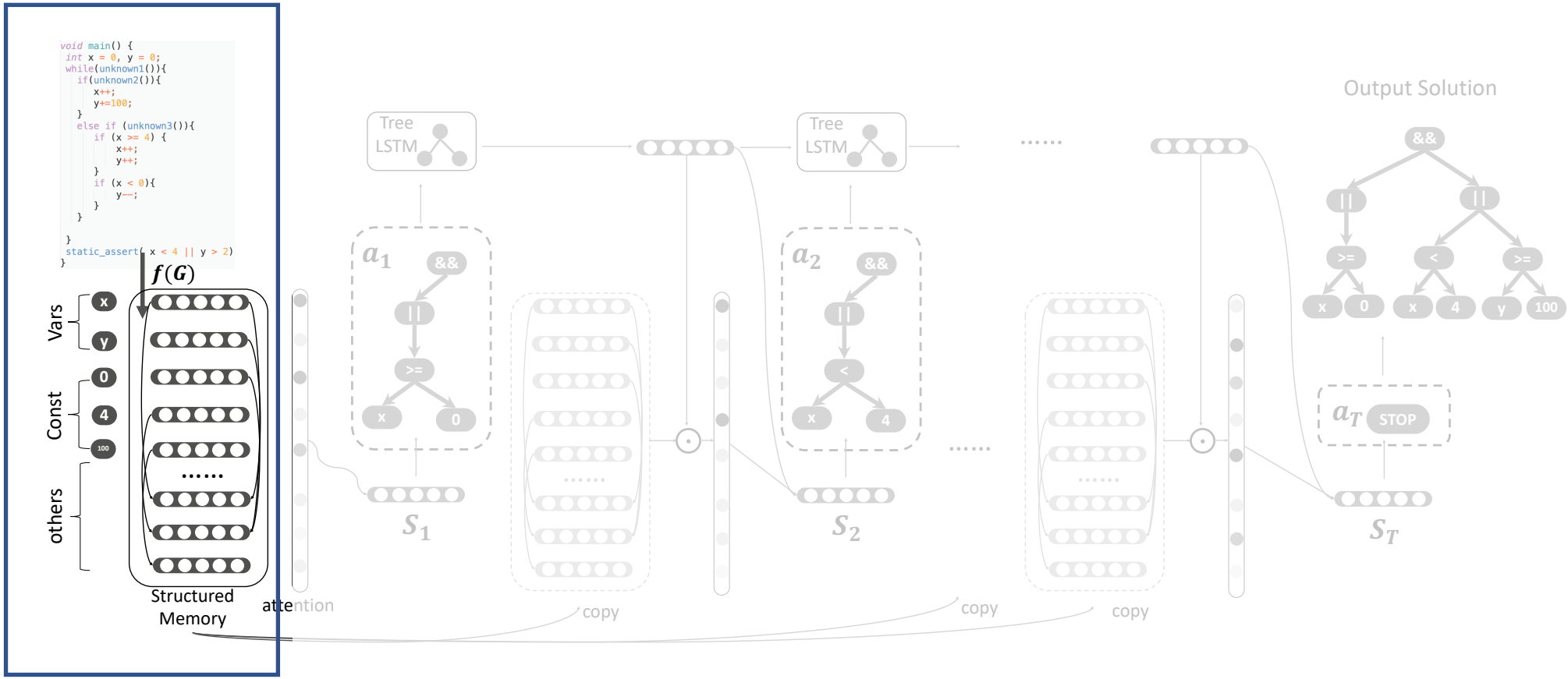
$$\forall x,y: \begin{cases} \texttt{true} \Rightarrow I[-50/x] & (pre) \\ I \wedge x<0 \Rightarrow I[(y+1)/y,(x+y)/x] & (inv) \\ I \wedge x \geq 0 \Rightarrow y>0 & (post) \end{cases}$$

Approach overview



Approach overview

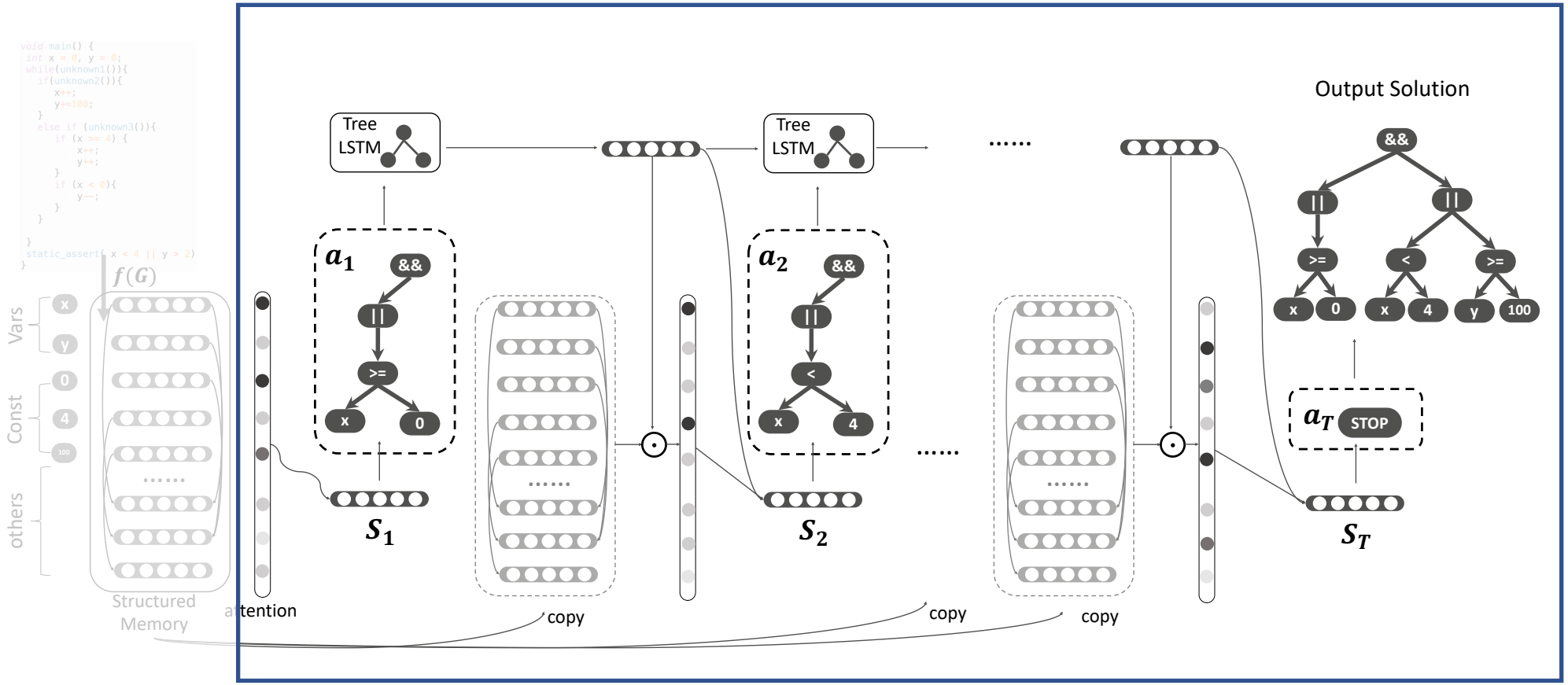
Representation Learning ”turn a program into a collection of high dimensional vectors”



Approach overview

Reinforcement Learning

”reduce loop invariant generation as a multiple step decision problem”



Representation learning for source code

```
while (y < 1000) {
    x = x + y
    y = y + 1
}
```

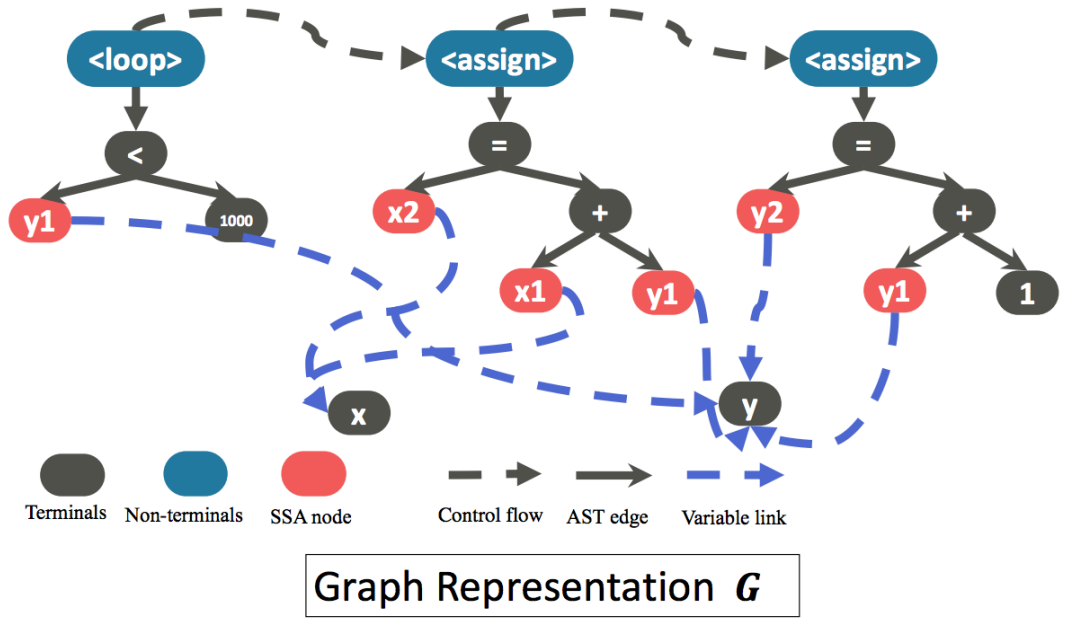
SSA

$x_1 = \phi(x_0, x_2)$
$y_1 = \phi(y_0, y_2)$

```
while (y1 < 1000) {
    x2 = x1 + y
    y2 = y1 + 1
}
```

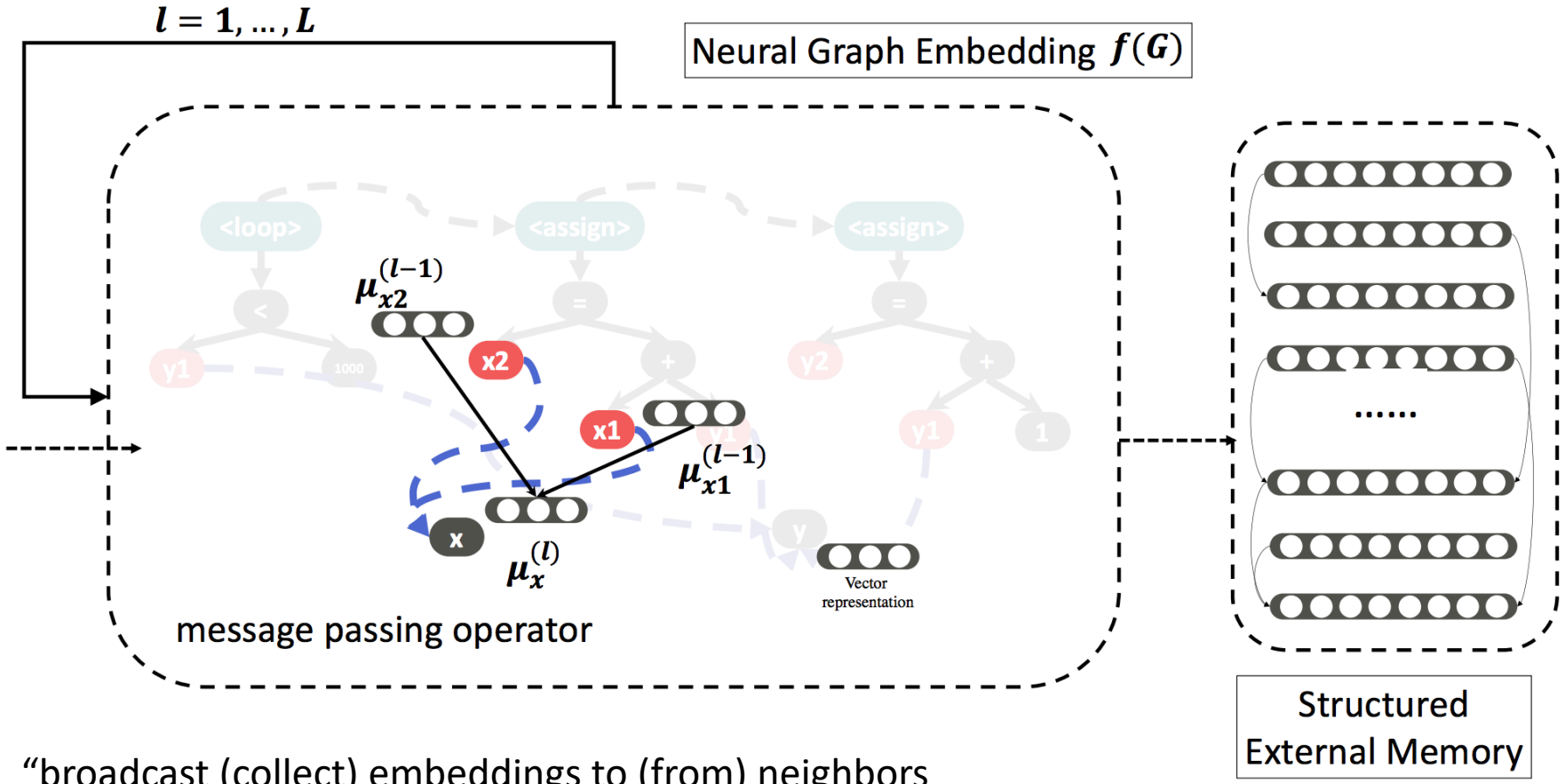
Source Code

```
while (y < 1000)
{
  x  = (x + y);
  y  = (y + 1);
}
```



Graph Representation G

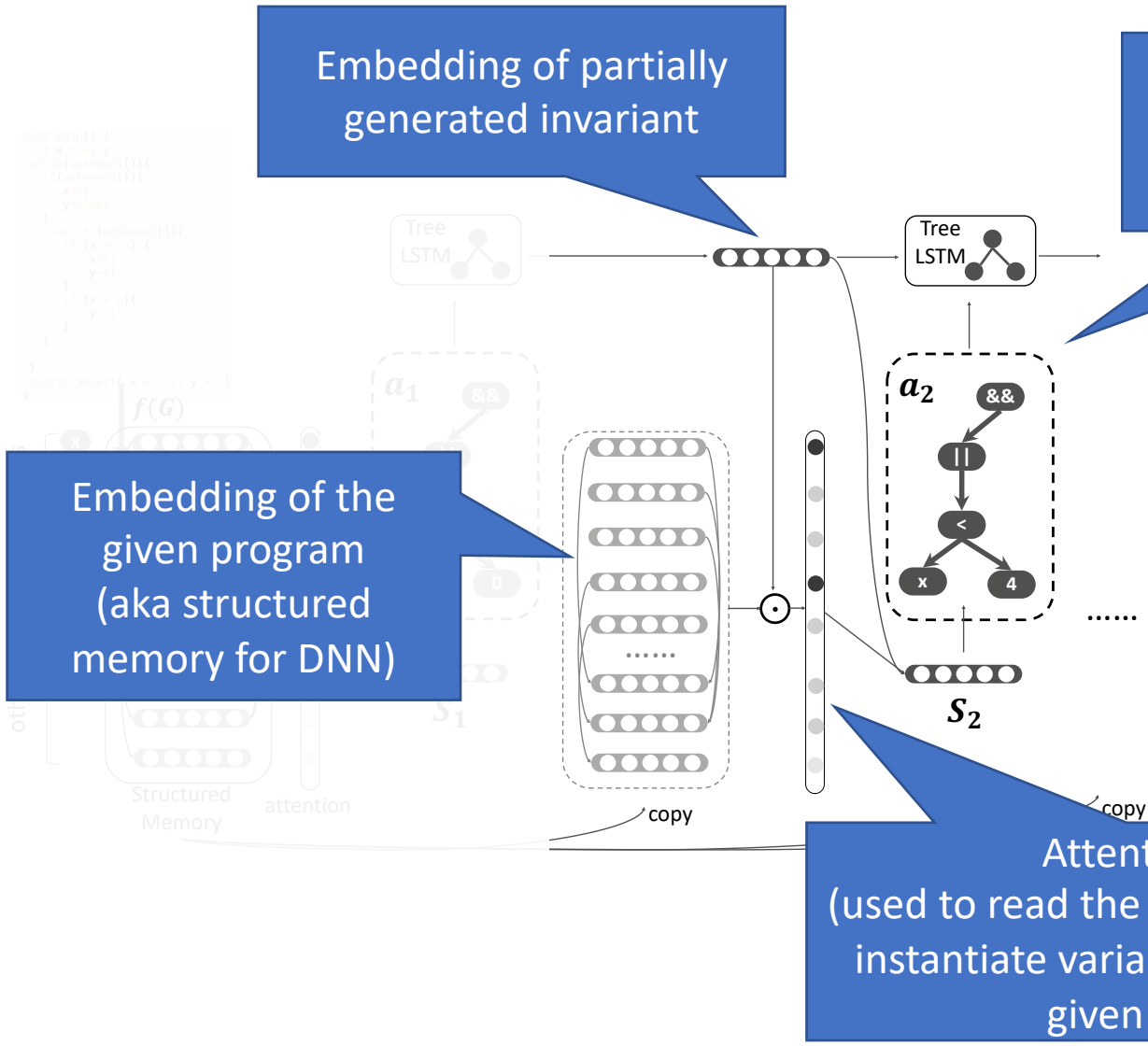
Representation learning for source code



"broadcast (collect) embeddings to (from) neighbors and perform non-linear transformation; repeat for L iterations"

"turn a piece of code into something readable by DNNs"

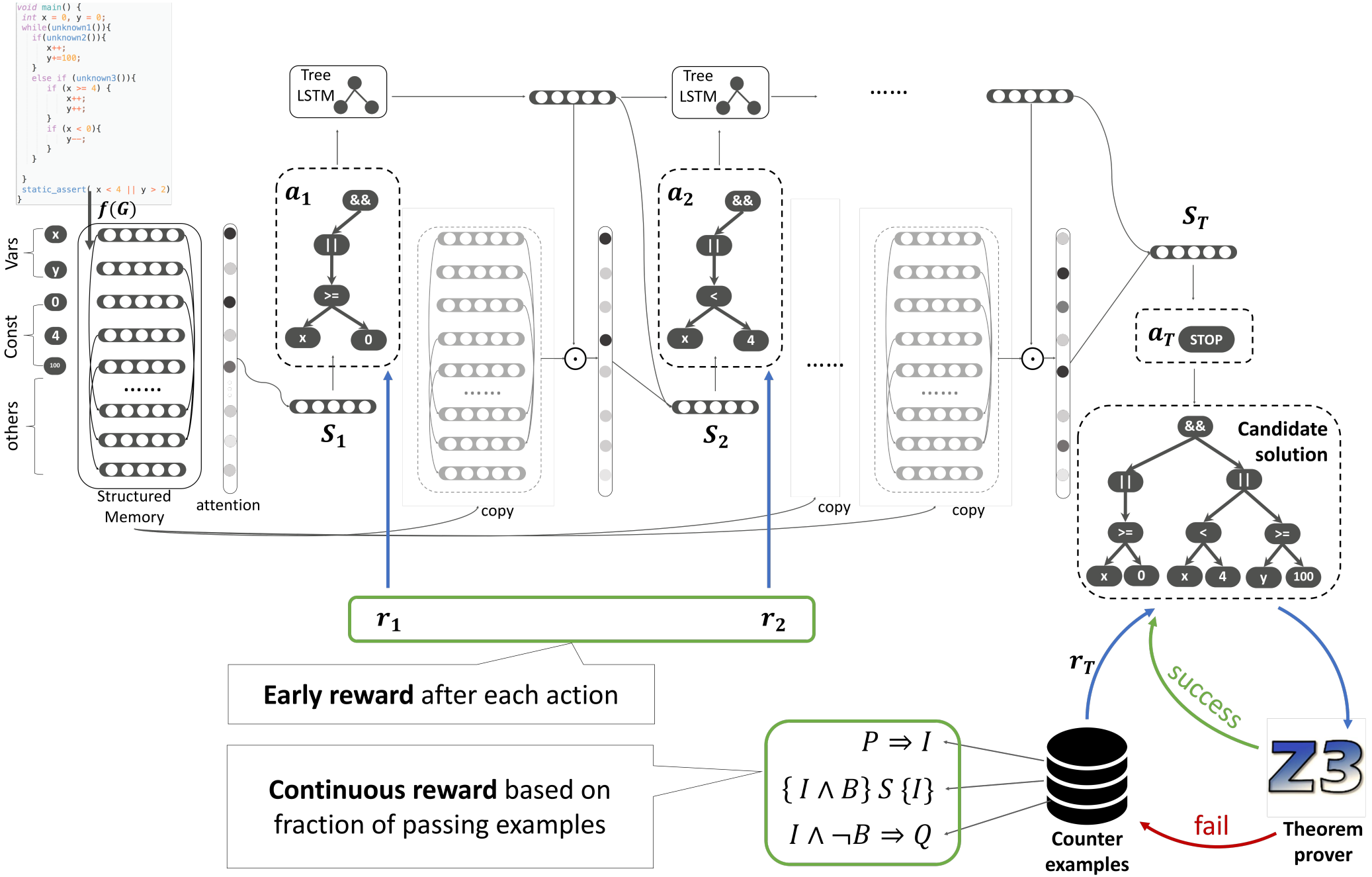
Reinforcement learning for invariant generation



P := P and P | P or P | E
E := e <= e | e < e | …
e := var | constant | e + e | e – e

Here, we are essentially learning the following distribution (which maximizes the reward discussed next)

$$\mathrm{P}(pred \mid program, partial\ invariant)$$

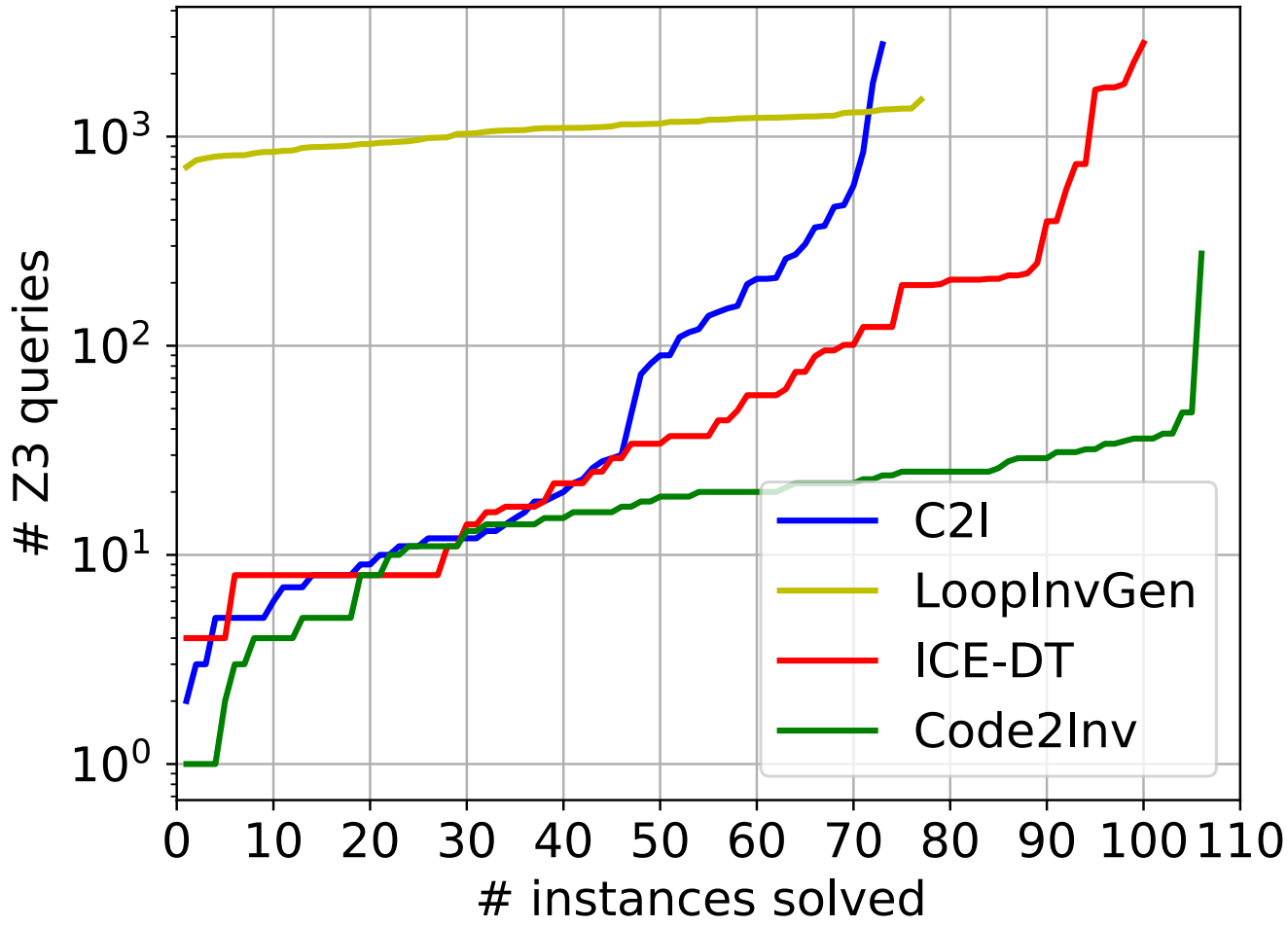

void main() {
int x = 0, y = 0;
while(unknown1()){
if(unknown2()){
x++;
y+=100;
}
else if (unknown3()){
if (x >= 4) {
x++;
y++;
}
if (x < 0){
y--;
}
}
}
static_assert(x < 4 || y > 2)
}
$f(G)$
Vars
Const
others
x
y
0
4
100
Structured Memory
attention
Tree LSTM
Tree LSTM
.....
a_1
a_2
&&
||
>=
<
x
0
4
S_1
S_2
copy
copy
copy
S_T
a_T
STOP
Candidate solution
r_1
r_2
Early reward after each action
Continuous reward based on fraction of passing examples
$P \Rightarrow I$
$\{I \wedge B\}\, S\, \{I\}$
$I \wedge \neg B \Rightarrow Q$
r_T
success
fail
Counter examples
Theorem prover
Z3

Experimental evaluation

- We collect 133 benchmark programs

OOPSLA 2013, Dillig et al

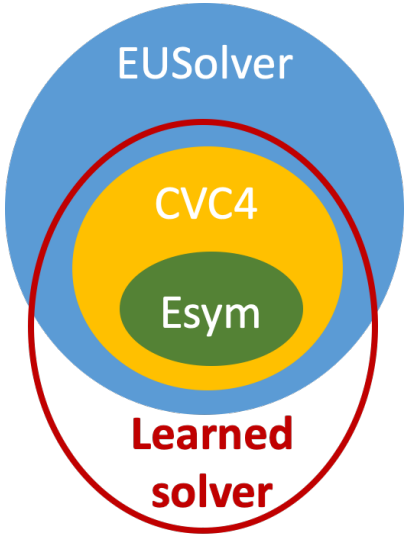
POPL 2016, Garag et al



Code and data: https://github.com/petablox/code2inv

Extension to program synthesis

- View a synthesis specification as a “program”
- Invariant generation is essentially program generation
- Initial results on 214 SyGuS tasks look promising



Thank you!

Q: how is the performance in term of running time?

- The running time for each solved instance takes up to 6 hours
  - All solvers have 6-hour limit (though other solvers tend to either solve an instance within 30 minutes or time out)
- Everything is done with a single thread CPU
- There is ***no training***, that is, each instance is solved from scratch (with randomly initialized weights)
  - View DRL as a smart search algorithm that evolves on the fly

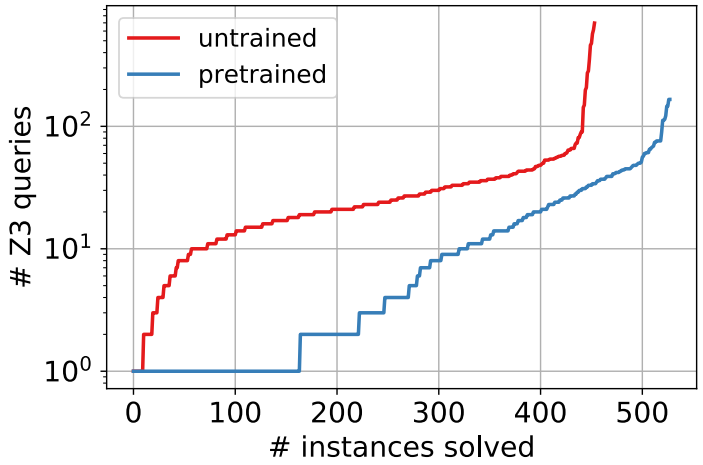
Q: how about the generation (suppose you do perform some pre-training)?

- We do not observe much generation across the collected benchmark, as they seem quite different from one to another.
  - Thus, pre-training does not help much
- We do have a generation study (see next two slides)

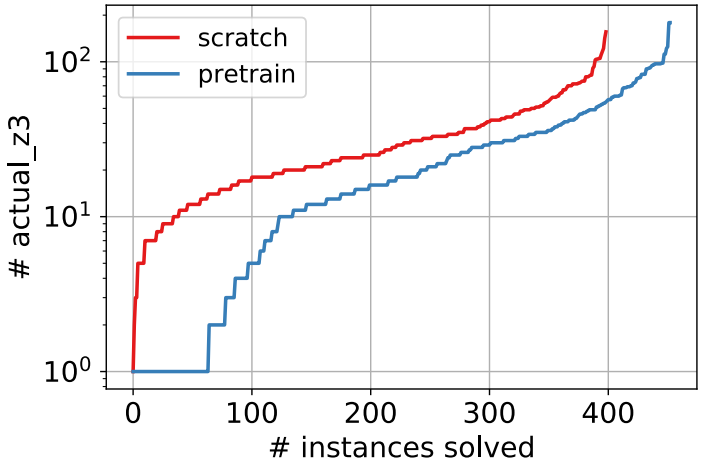
Generation study (injecting random statements)


```
void main (int n) {
    int x = 0
    int w = 0
    int m = 0
    int z = 0
    while (x < n) {
        z = z + 1
        if (unknown()) {
            m = x
            z = m + 1
        }
        x = x + 1
        w = m + x
    }
    if (n > 0) {
        assert (m < n)
    }
}
```

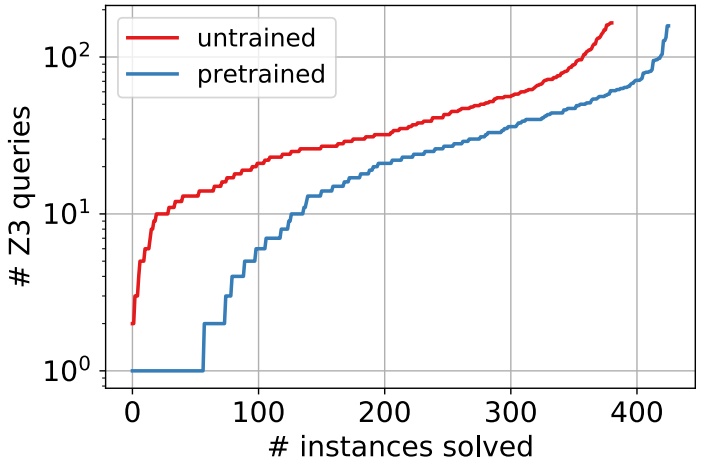
Generation study (evaluation)



1 confounding variable



3 confounding variables



5 confounding variables