

# Parametric Visual Program Induction

A Function Modularization Solution and Monto-Carlo Tree-Search Learning

Authors: **Xuguang Duan**, Xin Wang, Ziwei Zhang, Wenwu Zhu

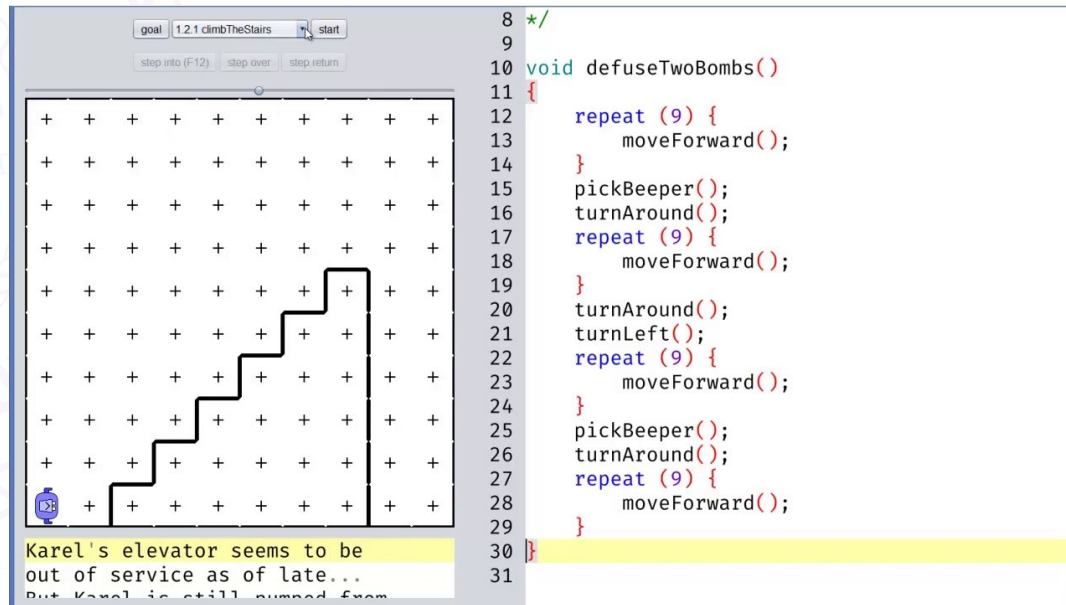
Department of Computer Science and Technology, Tsinghua University

Contact: [duan\\_xg@outlook.com](mailto:duan_xg@outlook.com); [xin\\_wang@tsinghua.edu.cn](mailto:xin_wang@tsinghua.edu.cn);

# Introduction

## Program Induction, or Program Learning..

aims to generate a program to describe the underlying logic or patterns:



The screenshot shows the Karel programming environment. On the left is a grid world with a staircase. On the right is a code editor with the following code:

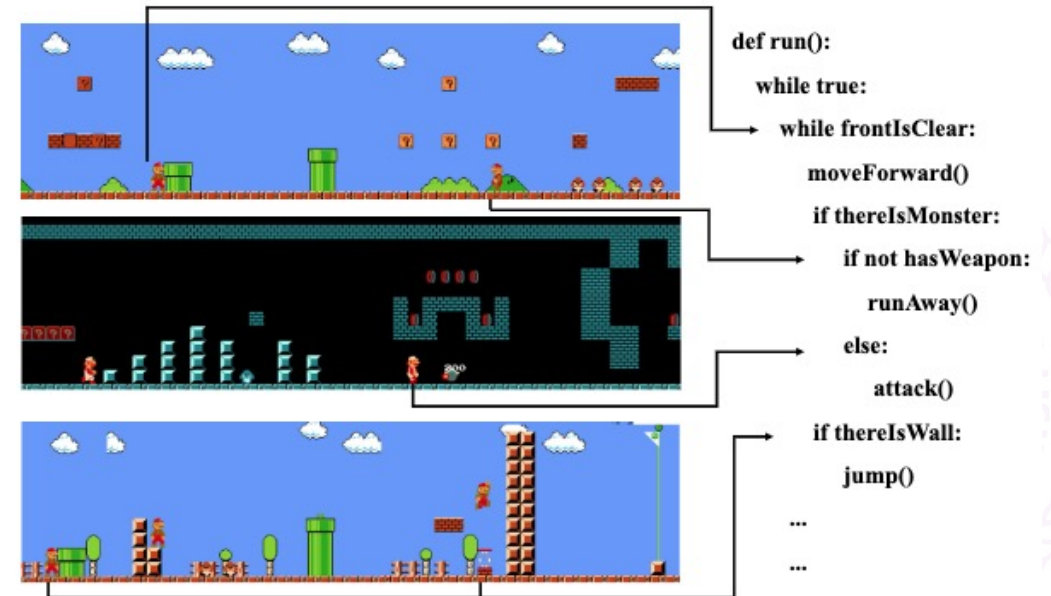
```

8 */
9
10 void defuseTwoBombs()
11 {
12     repeat (9) {
13         moveForward();
14     }
15     pickBeeper();
16     turnAround();
17     repeat (9) {
18         moveForward();
19     }
20     turnAround();
21     turnLeft();
22     repeat (9) {
23         moveForward();
24     }
25     pickBeeper();
26     turnAround();
27     repeat (9) {
28         moveForward();
29     }
30 }
31

```

Below the grid, there is a text box that says: "Karel's elevator seems to be out of service as of late... but Karel is still bumped from..."

(a) A program to control the *Karel* “robot”.



The screenshot shows three panels of Super Mario Bros. game levels. Arrows point from specific elements in the levels to a code editor on the right. The code is as follows:

```

def run():
while true:
while frontIsClear:
moveForward()
if thereIsMonster:
if not hasWeapon:
runAway()
else:
attack()
if thereIsWall:
jump()
...
...

```

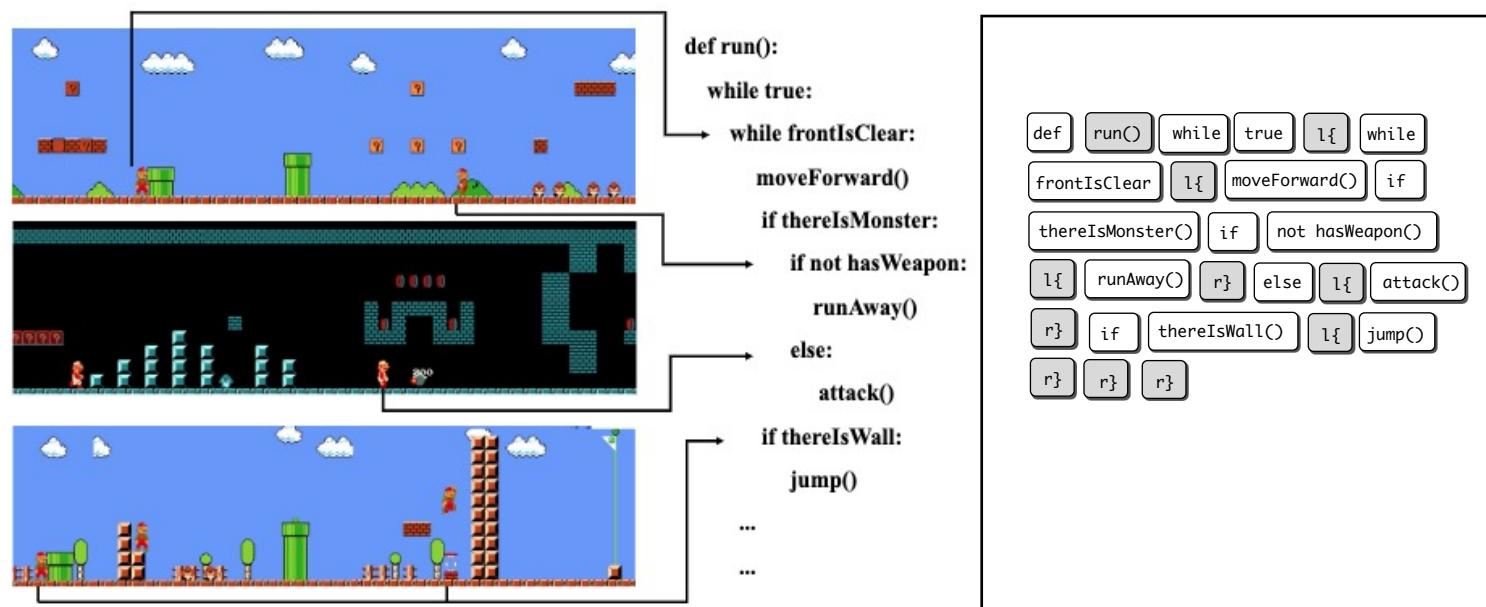
(b) A program to describe the logic under *Super Mario*.

# Introduction

## Program Induction, or Program Learning..

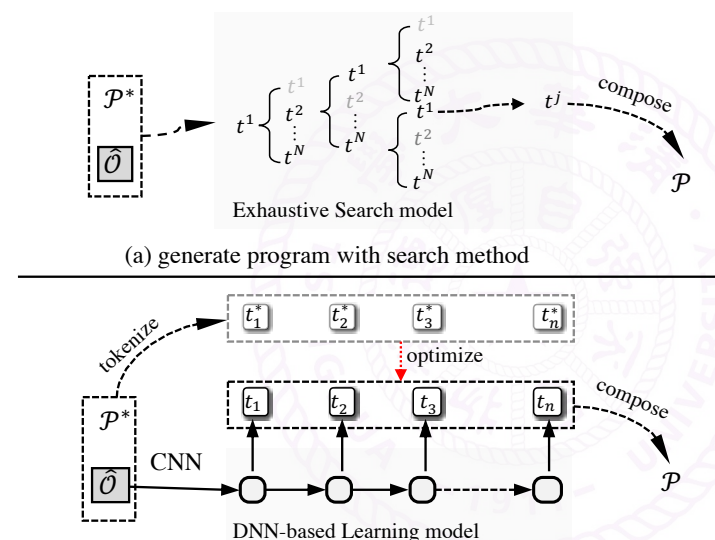
is usually solved via program tokenization and then **Learning** or/and **Searching**.

program tokenization: split a program into symbolic tokens:



(I) Super Mario and its game logic;

(II) tokenized program;



(a) generate program with search method

(b) generate program with learning model

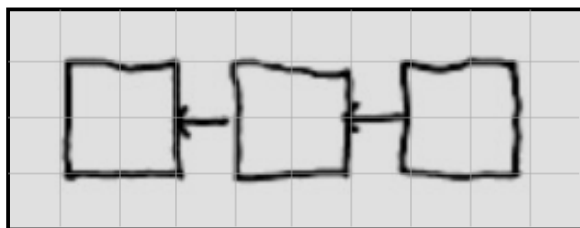
(III) how to generate a proper program.

# Introduction

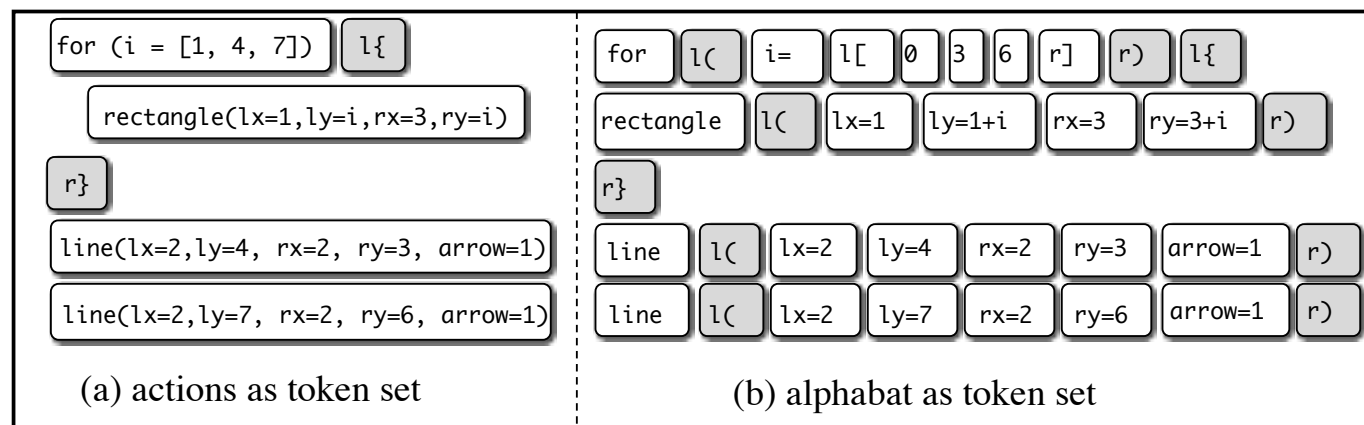
## However, When it comes to Visual Program Induction:

two challenges raises due to the huge program space...

Firstly, the modeling of functions:



```
for (i = [0, 3, 6]){
    rectangle(lx=1,ly=1+i,rx=3,ry=3+i);
}
line(lx=2,ly=4, rx=2, ry=3, arrow=1);
line(lx=2,ly=7, rx=2, ry=6, arrow=1);
```



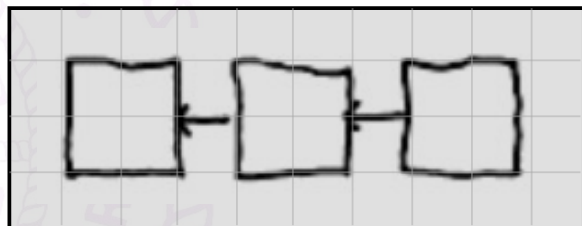
We may obtain a huge token set by **actions as token set** or we have to deal with the fragile program syntax with **alphabat as token set**

# Introduction

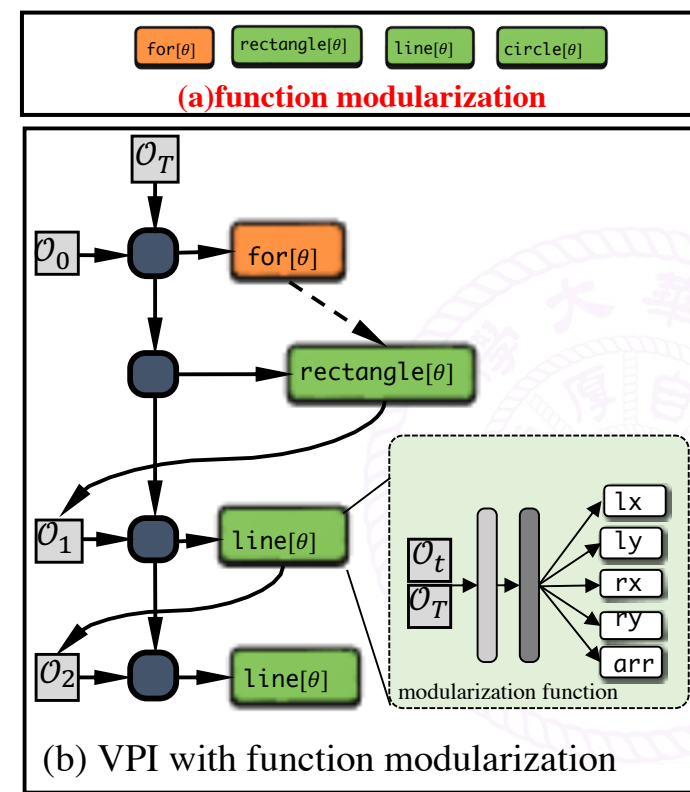
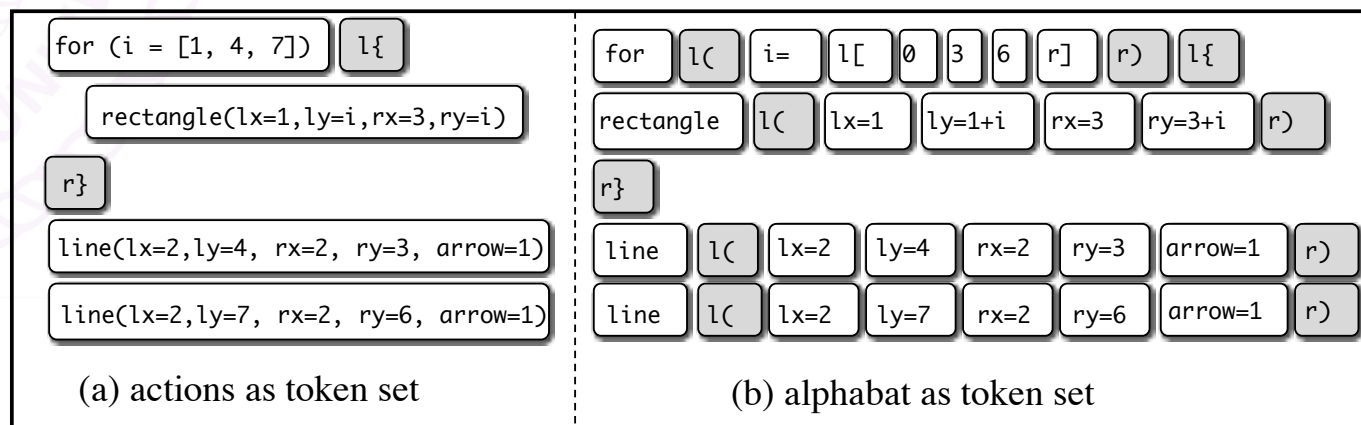
However, When it comes to Visual Program Induction:

two challenges raises due to the huge program space...

Firstly, the modeling of functions:



```
for (i = [0, 3, 6]){
    rectangle(lx=1,ly=1+i,rx=3,ry=3+i);
}
line(lx=2,ly=4, rx=2, ry=3, arrow=1);
line(lx=2,ly=7, rx=2, ry=6, arrow=1);
```

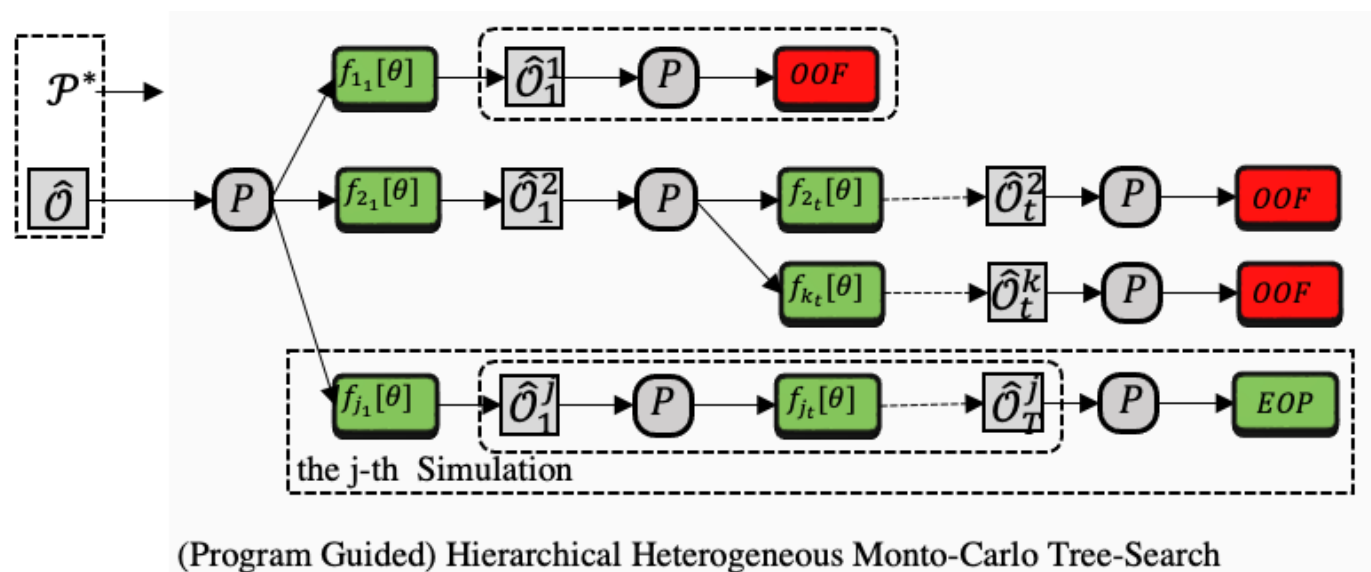
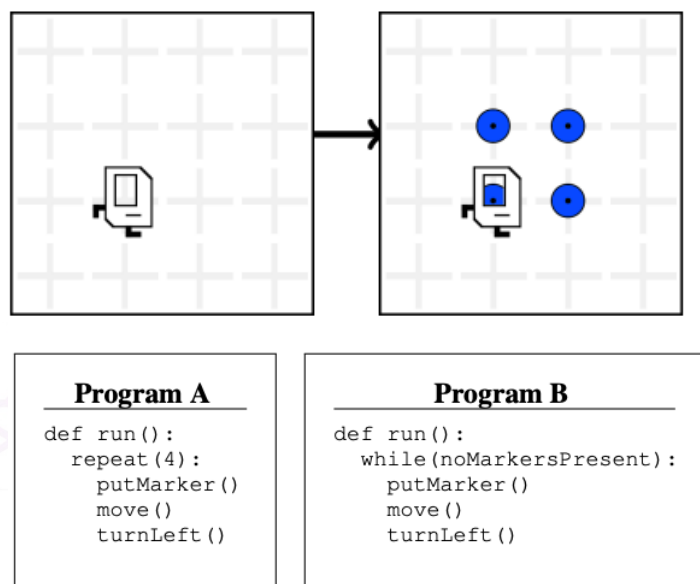


By using function modules; we could separately learn the function dynamics and function parameters.

# Method

However, When it comes to Visual Program Induction:  
two challenges raises due to the huge program space...

Secondly, the learning of programs:



(I) The program alias;

(II) Our solution of Monto-Carlo Tree-Search solution.

# Experiment

We test our methods on several datasets:

firstly, the Pixel-Grid dataset: function tokenization vs. function modules

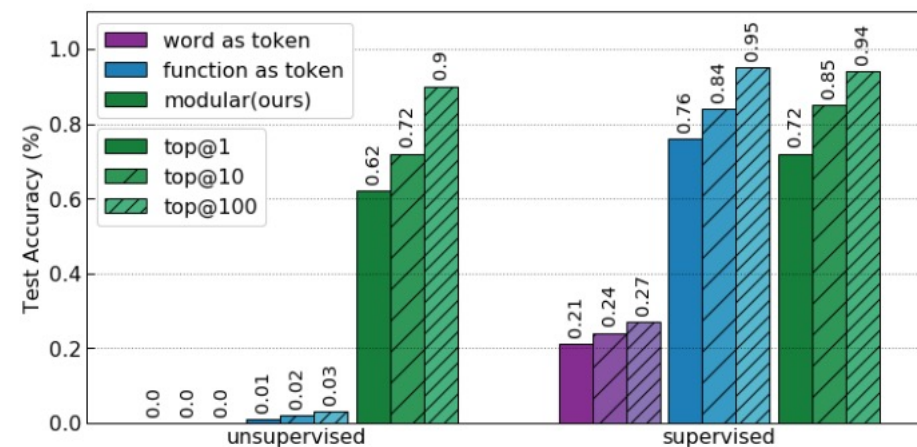
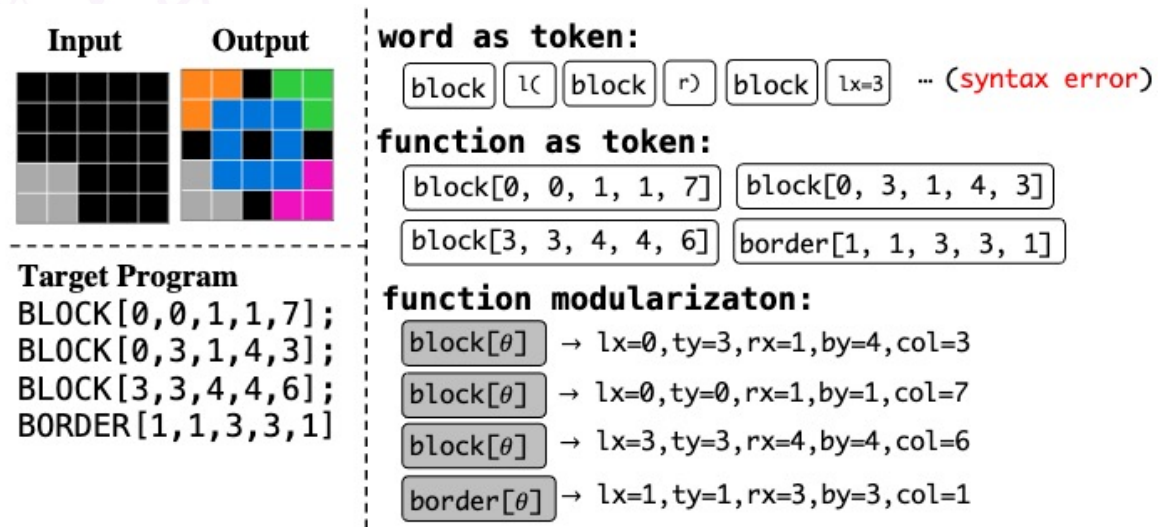
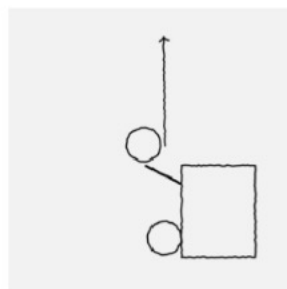


Figure 3. The testing accuracy (%) on the  $5 \times 5$  Pixel Grid dataset.

# Experiment

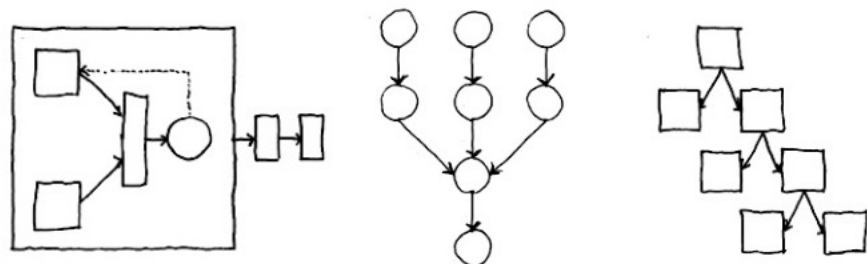
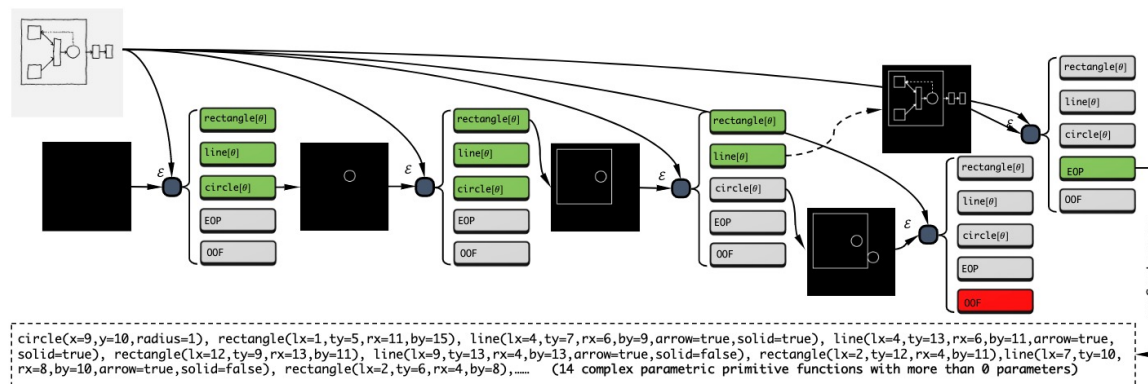
We test our methods on several datasets:

secondly, the Latex-Drawing dataset: Control-free Program Learning

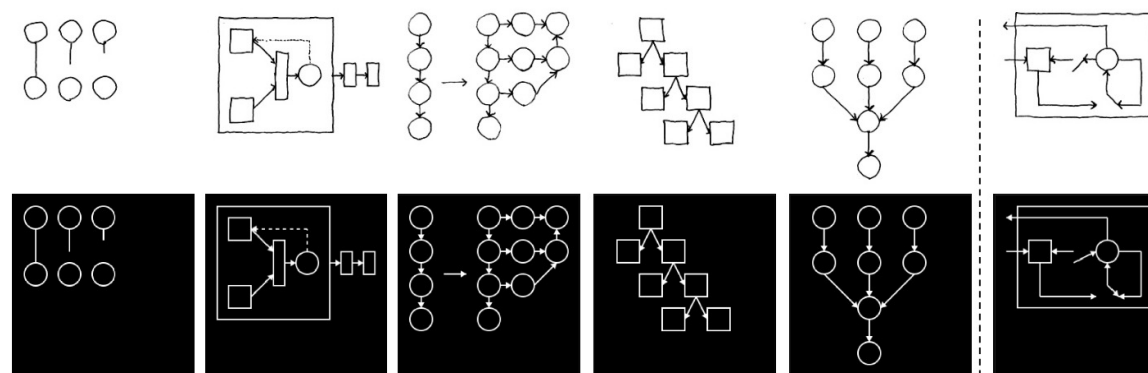


```
circle(x=8,y=8,radius=1),
circle(x=9,y=14, radius=1),
line(x1=9,y1=8,x2=9,y2=1,
      arrow=true,solid=true),
line(x1=8,y1=9,x2=10,y2=10,
      arrow=false,solid=true),
rectangle(x1=10,y1=9,x2=15,y2=15)
```

(a) synthesized observation and its ground truth program.



(b) hand drawn images for testing.





# Experiment

We test our methods on several datasets:

third, the 3D Shape dataset: Control-based Program Learning

(Control Transition)  $C : (\mathcal{O}_i, \mathcal{O}_O) \rightarrow C_i$ .

(Function Transition)  $P : (\mathcal{O}_i, \mathcal{O}_O) \rightarrow f_i$ ,

(Parameter Prediction)  $Q_f : (\mathcal{O}_i, \mathcal{O}_O, C_i) \rightarrow \Theta$ .



observations



rendered 3D object ← learned program

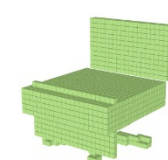
```
draw('Top', 'Rec', P=(-1,-1,0), G=(2,7,8))
draw('Sup', 'Cylinder', P=(-11,0,0), G=(11,1))
for(i<4, 'Rot', theta=90°, axis=(-12,0,0))
  draw('Base', 'Line', P1=(-12,0,0), P2=(-12,-7,-7),
  theta*i, axis)
draw('Back', 'Cub', P=(1,5,-7), G=(11,2,14), theta=10°)
for(i<2, 'Trans', u=(0,0,13))
  draw('Chair_Beam', 'Cub', P=(1,-3,-7)+i*u, G=(3,1,1))
for(i<2, 'Trans', u=(0,0,14))
  draw('Hori_Bar', 'Cub', P=(4,-3,-7)+i*u, G=(2,8,1))
```



```
draw('Top', 'Rec', P=(-1,0,0), G=(3,9,9))
for(i<2, 'Trans', u1=(0,0,17))
  for(i<2, 'Trans', u2=(0,13,0))
    draw('Leg', 'Cub', P=(-12,-7,-9)+i*u1+j*u2, G=(14,2,1))
for(i<2, 'Trans', u=(0,0,17))
  draw('Hori_Bar', 'Cub', P=(-12,-7,-9)+i*u, G=(2,15,1))
draw('Back', 'Cub', P=(2,5,-9), G=(10,3,18), theta=5°)
for(i<2, 'Trans', u=(0,0,18))
  draw('Chair_Beam', 'Cub', P=(2,-7,-10)+i*u, G=(3,1,1))
for(i<2, 'Trans', u=(0,0,18))
  draw('Hori_Bar', 'Cub', P=(5,-7,-10)+i*u, G=(3,14,1))
```

```
draw('Top', 'Square', P=(10,0,0), G=(3,12))
for(i<2, 'Trans', u1=(0,0,16))
  for(i<2, 'Trans', u2=(0,17,0))
    draw('Leg', 'Cub', P=(-12,-10,-9)+i*u1+j*u2,
    G=(24,3,2))
draw('Layer', 'Rec', P=(-2,0,0), G=(2,9,9))
```

```
draw('Top', 'Square', P=(-5,0,0), G=(5,10))
draw('Vert_Board', 'Cub', P=(-10,-8,-10), G=(11,1,19))
for(i<5, 'Rot', theta=72°, axis=(-11,1,0))
  draw('Base', 'Line', P1=(-11,1,0), P2=(-12,-8,-6), theta*i, axis)
draw('Back', 'Cub', P=(0,10,-10), G=(11,2,19), theta=0°)
```



# Thanks

