Supplement to: Inferring Graphics Programs from Images

Anonymous Author(s)
Affiliation
Address
email

1 Neural network architecture

1.1 Convolutional network

The convolutional network takes as input 2 256×256 images represented as a $2 \times 256 \times 256 \times$ volume. These are passed through two layers of convolutions separated by ReLU nonlinearities and max pooling:

- Layer 1: 20 8×8 convolutions, 2 16×4 convolutions, 2 4×16 convolutions. Followed by 8×8 pooling with a stride size of 4.
- Layer 2: 10 8×8 convolutions. Followed by 4×4 pooling with a stride size of 4.

Training takes a little bit less than a day on a Nvidia TitanX GPU. The network was trained on 10^5 synthetic examples.

1.2 Autoregressive decoding of drawing commands

Given the image features f, we predict the first token using logistic regression:

$$\mathbb{P}[T_1] \propto W_{T_1} f \tag{1}$$

where W_{T_1} is a learned weight matrix.

Subsequent tokens are predicted as:

$$\mathbb{P}[T_n | T_{1:(n-1)}] \propto \text{MLP}_{T_1,n}(I \otimes \bigotimes_{j<n} \text{oneHot}(T_j)) \tag{2}$$

Thus each token of each drawing primitive has its own learned MLP. For predicting the coordinates of lines we found that using 32 hidden nodes with sigmoid activations worked well; for other tokens the MLP's are just logistic regression (no hidden nodes).

1.3 A learned likelihood surrogate

Our architecture for $L_{\text{learned}}(\text{render}(T_1)|\text{render}(T_2))$ has the same series of convolutions as the network that predicts the next drawing command. We train it to predict two scalars: $|T_1 - T_2|$ and $|T_2 - T_1|$. These predictions are made using linear regression from the image features followed by a ReLU nonlinearity; this nonlinearity makes sense because the predictions can never be negative but could be arbitrarily large positive numbers.

We train this network by sampling random synthetic scenes for T_1, and then perturbing them in small ways to produce T_2. We minimize the squared loss between the network's prediction and the ground truth symmetric differences. T_1 is rendered in a "simulated hand drawing" style which we describe next.

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

2 Simulating hand drawings

We introduce noise into the LaTeX rendering process by:

- Rescaling the image intensity by a factor chosen uniformly at random from $[0.5, 1.5]$
- Translating the image by ± 3 pixels chosen uniformly random
- Rendering the LaTeX using the `pencildraw` style, which adds random perturbations to the paths drawn by LaTeXin a way designed to resemble a pencil.
- Randomly perturbing the positions and sizes of primitive LaTeXdrawing commands

3 Likelihood surrogate for synthetic data

For synthetic data (e.g., LaTeX output) it is relatively straightforward to engineer an adequate distance measure between images, because it is possible for the system to discover drawing commands that exactly match the pixels in the target image. We use:

$$-\log L(I_1|I_2) = \sum_{1\leq x\leq 256} \sum_{1\leq y\leq 256} |I_1[x,y] - I_2[x,y]| \begin{cases} \alpha, \text{if } \mathrm{I}_1[x,y] > I_2[x,y] \\ \beta, \text{if } \mathrm{I}_1[x,y] < I_2[x,y] \\ 0, \text{if } \mathrm{I}_1[x,y] = I_2[x,y] \end{cases} \tag{3}$$

where α, β are constants that control the trade-off between preferring to explain the pixels in the image (at the expense of having extraneous pixels) and not predicting pixels where they don't exist (at the expense of leaving some pixels unexplained). Because our sampling procedure incrementally constructs the scene part-by-part, we want $\alpha > \beta$. That is, it is preferable to leave some pixels unexplained; for once a particle in SMC adds a drawing primitive to its trace that is not actually in the latent scene, it can never recover from this error. In our experiments on synthetic data we used $\alpha = 0.8$ and $\beta = 0.04$.

4 Generating synthetic training data

We generated synthetic training data for the neural network by sampling LaTeX code according to the following generative process: First, the number of objects in the scene are sampled uniformly from 1 to 8. For each object we uniformly sample its identity (circle, rectangle, or line). Then we sample the parameters of the circles, than the parameters of the rectangles, and finally the parameters of the lines; this has the effect of teaching the network to first draw the circles in the scene, then the rectangles, and finally the lines. We furthermore put the circle (respectively, rectangle and line) drawing commands in order by left-to-right, bottom-to-top; thus the training data enforces a canonical order in which to draw any scene.

To make the training data look more like naturally occurring figures, we put a Chinese restaurant process prior **?** over the values of the X and Y coordinates that occur in the execution trace. This encourages reuse of coordinate values, and so produces training data that tends to have parts that are nicely aligned.

In the synthetic training data we excluded any sampled scenes that had overlapping drawing commands. As shown in the main paper, the network is then able to generalize to scenes with, for example, intersecting lines or lines that penetrate a rectangle.

When sampling the endpoints of a line, we biased the sampling process so that it would be more likely to start an endpoint along one of the sides of a rectangle or at the boundary of a circle. If n is the number of points either along the side of a rectangle or at the boundary of a circle, we would sample an arbitrary endpoint with probability $\frac{2}{2+n}$ and sample one of the "attaching" endpoints with probability $\frac{1}{2+n}$.

See figure **??** for examples of the kinds of scenes that the network is trained on.

For readers wishing to generate their own synthetic training sets, we refer them to our source code at: `http://www.redactedForAnonymousReview.com`.

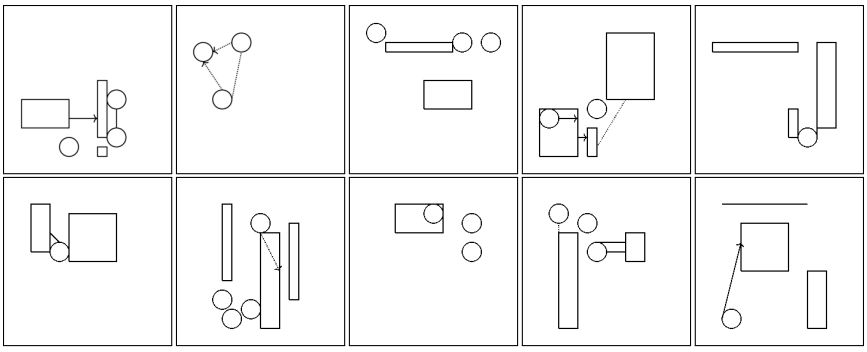


Figure 1: Example synthetic training data

5 The cost function for programs

We seek the minimum cost program which evaluates to (produces the drawing primitives in) an execution trace T:

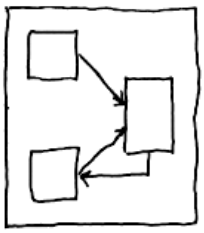
$$\text{program}(T) = \underset{\substack{p \in \text{DSL} \\ p \text{ evaluates to } T}}{\arg\min} \ \text{cost}(p) \tag{4}$$

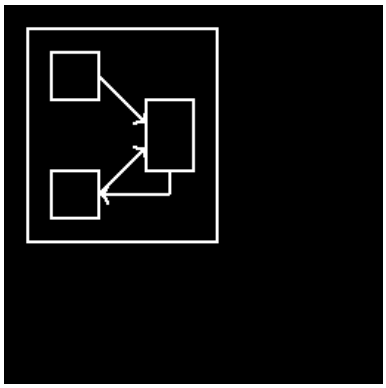
Programs incur a cost of 1 for each command (primitive drawing action, loop, or reflection). They incur a cost of $\frac{1}{3}$ for each unique coefficient they use in a linear transformation beyond the first coefficient. This encourages reuse of coefficients, which leads to code that has translational symmetry; rather than provide a translational symmetry operator as we did with reflection, we modify what is effectively a prior over the space of program so that it tends to produce programs that have this symmetry.

Programs also incur a cost of 1 for having loops of constant length 2; otherwise there is often no pressure from the cost function to explain a repetition of length 2 as being a reflection rather a loop.

6 Full results on drawings data set

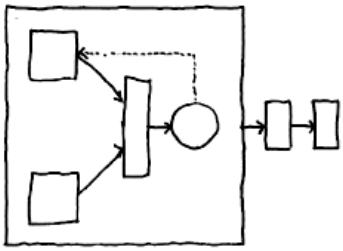
Below we show our full data set of drawings. The leftmost column is a hand drawing. The middle column is a rendering of the most likely trace discovered by the neurally guided SMC sampling scheme. The rightmost column is the program we synthesized from a ground truth execution trace of the drawing.

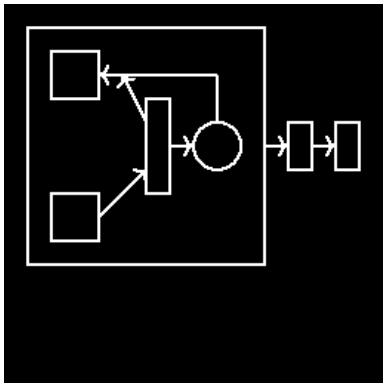




```
line(6,2,6,3,
arrow = False,solid = True);
line(6,2,3,2,
arrow = True,solid = True);
reflect(reflect(y = 9)){
line(3,2,5,4,
arrow = True,solid = True);
rectangle(0,0,8,9);
rectangle(5,3,7,6);
rectangle(1,1,3,3)
}
```

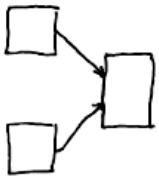
88



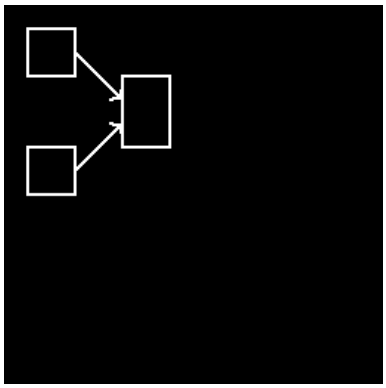


Solver timeout

89

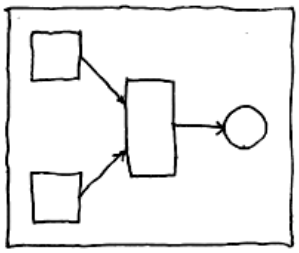


90

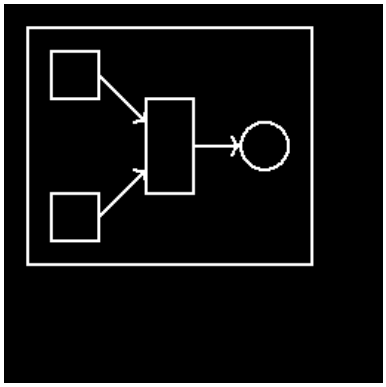


```
rectangle(4,2,6,5);
reflect(reflect(y = 7)){
line(2,6,4,4,
arrow = True,solid = True);
rectangle(0,0,2,2)
}
```

91



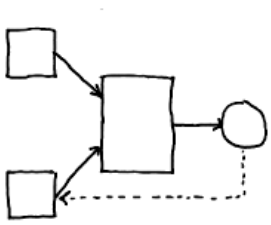
92

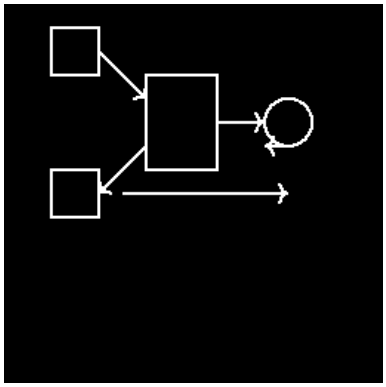


```
circle(10,5);
line(7,5,9,5,
arrow = True,solid = True);
rectangle(5,3,7,7);
rectangle(0,0,12,10);
reflect(reflect(y = 10)){
line(3,8,5,6,
arrow = True,solid = True);
rectangle(1,7,3,9)
}
```

93

94

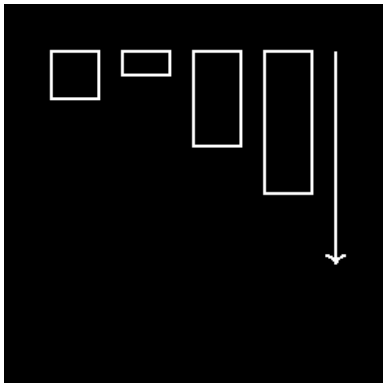




```
circle(10,4);
line(10,1,2,1,
arrow = True,solid = False);
line(10,1,10,3,
arrow = False,solid = False);
line(7,4,9,4,
arrow = True,solid = True);
reflect(reflect(y = 8)){
line(2,7,4,5,
arrow = True,solid = True);
rectangle(4,2,7,6);
rectangle(0,6,2,8)
}
```

95

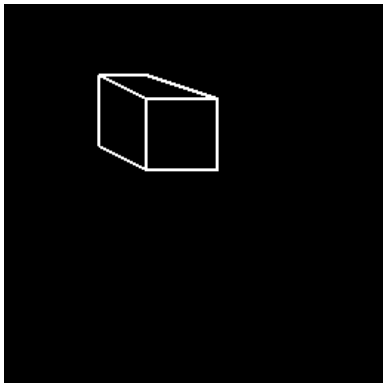
96



97

Solver timeout

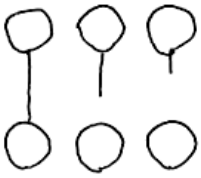
98

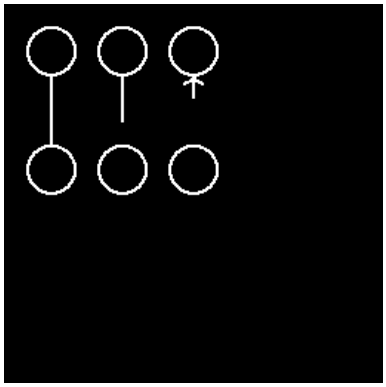


```
line(0,1,2,0,
arrow = False,solid = True);
for (i < 3){
if (i > 0){
line(3*i + -3,4,3*i + -1,3,
arrow = False,solid = True);
line(0,3*i + -2,3*i + -3,4,
arrow = False,solid = True)
}
rectangle(2,0,5,3)
}
```

99

100

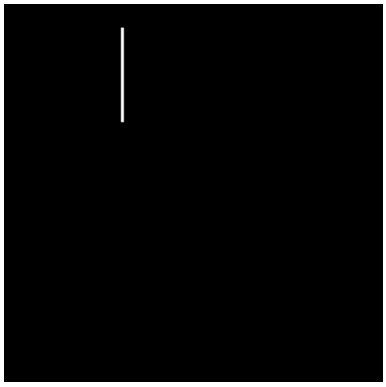




```
for (i < 3){
circle(-3*i + 7,1);
circle(-3*i + 7,6);
line(-3*i + 7,-1*i + 4,-3*i + 7,
arrow = False,solid = True)
}
```

101

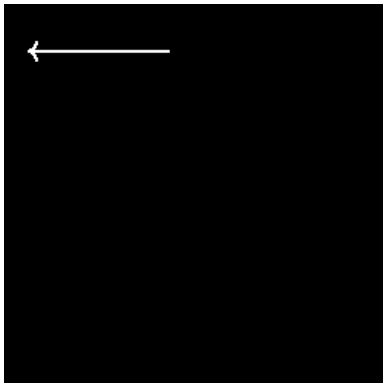
102



```
line(0,0,0,4,
arrow = False,solid = True)
```

103

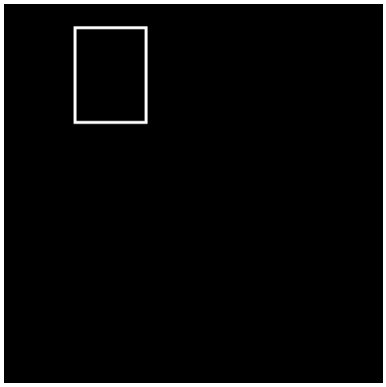
104



```
line(6,0,0,0,
arrow = True,solid = True)
```

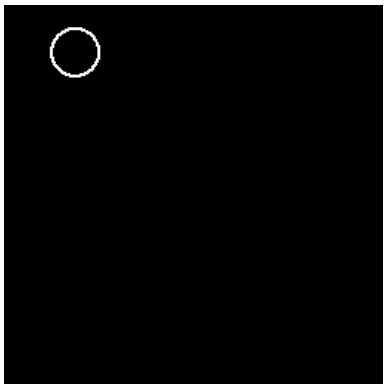
105

106



```
rectangle(0,0,3,4)
```

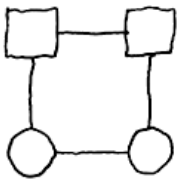
107

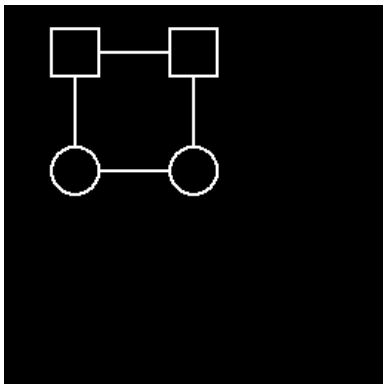


108

```
circle(1,1)
```

109

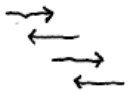




110

```
line(2,6,5,6,
arrow = False,solid = True);
reflect(reflect(x = 7)){
circle(6,1);
line(2,1,5,1,
arrow = False,solid = True);
line(1,2,1,5,
arrow = False,solid = True);
rectangle(5,5,7,7)
}
```

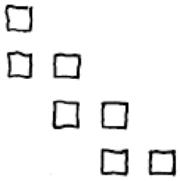
111

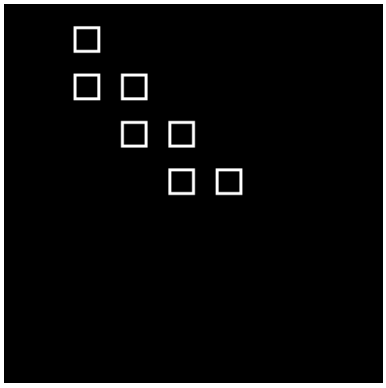


112

```
line(2,1,4,1,
arrow = True,solid = True);
line(3,2,1,2,
arrow = True,solid = True);
line(5,0,3,0,
arrow = True,solid = True);
line(0,3,2,3,
arrow = True,solid = True)
```

113

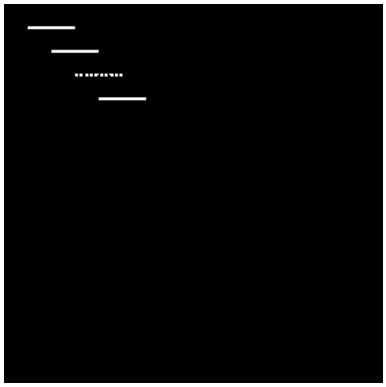




114

```
for (i < 4){
if (i > 0){
rectangle(-2*i + 6,2*i + -2,-2*i
}
rectangle(-2*i + 6,2*i,-2*i + 7,
}
```

115

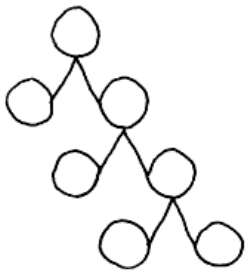


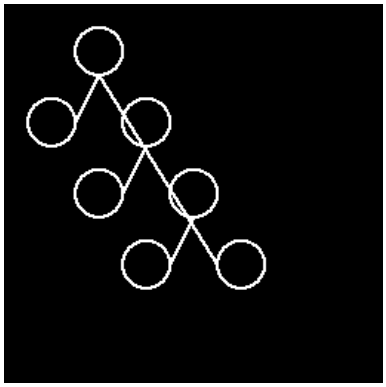
116

```
line(0,3,2,3,
arrow = False,solid = False);
line(2,1,4,1,
arrow = False,solid = False);
line(1,2,3,2,
arrow = False,solid = True);
line(3,0,5,0,
arrow = False,solid = True)
```

117

118

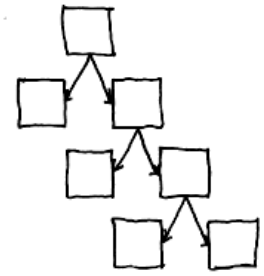


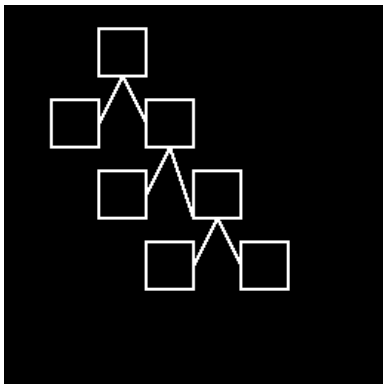


```
for (i < 4){
if (i > 0){
circle(-2*i + 7,3*i + -2);
line(-2*i + 9,3*i,-2*i + 10,3*i
arrow = False,solid = True);
line(-2*i + 8,3*i + -2,-2*i + 9,
arrow = False,solid = True)
}
circle(-2*i + 9,3*i + 1)
}
```

119

120

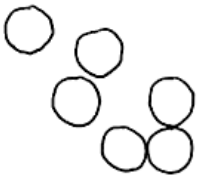


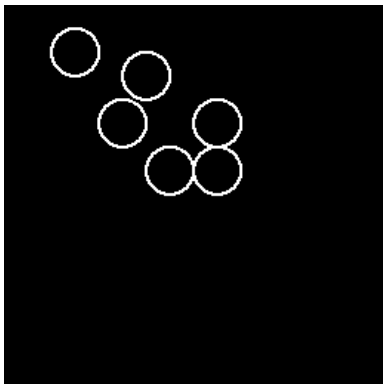


```
for (i < 4){
if (i > 0){
line(2*i + 1,-3*i + 12,2*i,-3*i
arrow = True,solid = True);
line(2*i + 1,-3*i + 12,2*i + 2,-
arrow = True,solid = True);
rectangle(2*i + -2,-3*i + 9,2*i,
}
rectangle(2*i + 2,-3*i + 9,2*i +
}
```

121

122

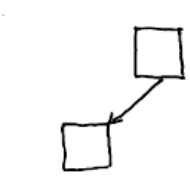


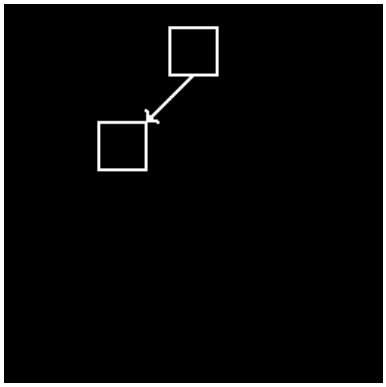


```
circle(5,1);
for (i < 3){
if (i > 0){
circle(7,2*i + -1);
circle(i + 2,2*i + 1)
}
circle(1,6)
}
```

123

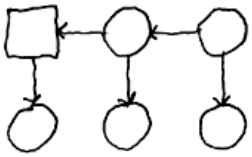
124

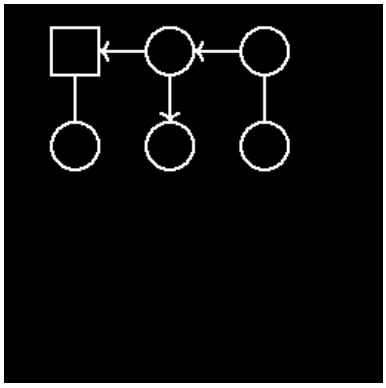




```
line(4,4,2,2,
arrow = True,solid = True);
rectangle(3,4,5,6);
rectangle(0,0,2,2)
```

125

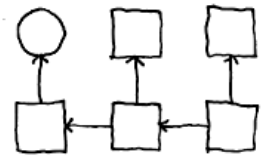


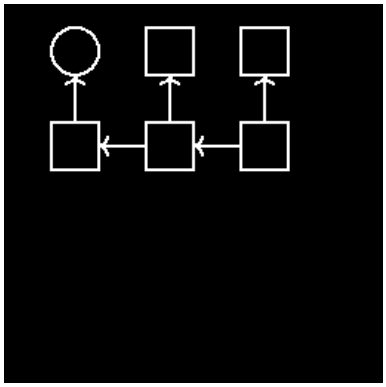


126

```
rectangle(0,4,2,6);
for (i < 3){
if (i > 0){
line(-4*i + 12,5,-4*i + 10,5,
arrow = True,solid = True);
for (j < i + 1){
circle(-4*j + 9,-4*i + 9)
}
}
line(-4*i + 9,4,-4*i + 9,2,
arrow = True,solid = True)
}
```

127

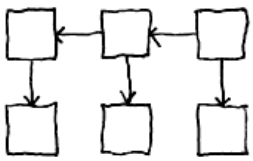




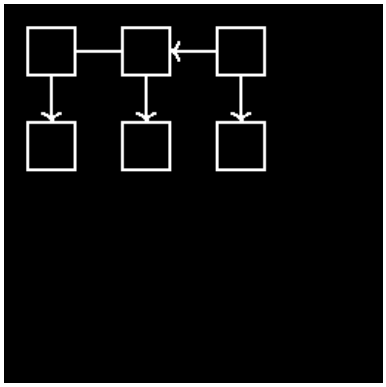
128

Solver timeout

129

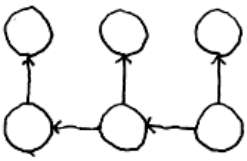


130

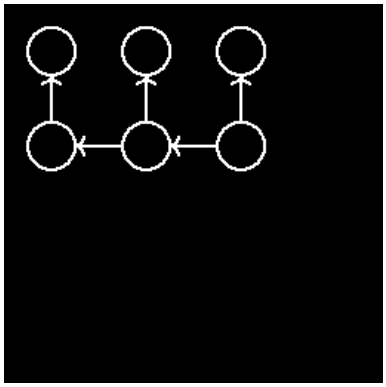


```
for (i < 3){
line(-4*i + 9,4,-4*i + 9,2,
arrow = True,solid = True);
for (j < 2){
line(-4*j + 8,5,-4*j + 6,5,
arrow = True,solid = True);
rectangle(-4*i + 8,4*j,-4*i + 10
}
}
```

131

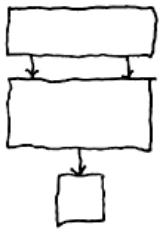


132

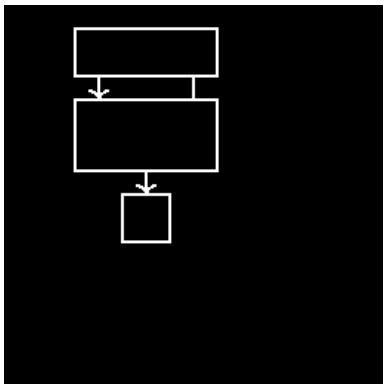


```
for (i < 3){
if (i > 0){
line(-4*i + 12,1,-4*i + 10,1,
arrow = True,solid = True)
}
circle(-4*i + 9,1);
circle(-4*i + 9,5);
line(-4*i + 9,2,-4*i + 9,4,
arrow = True,solid = True)
}
```

133



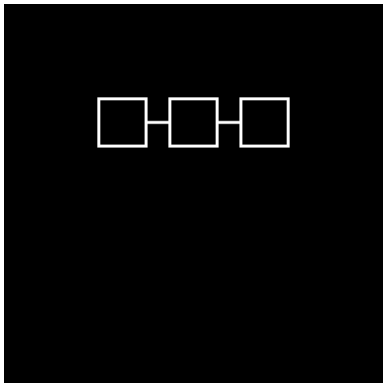
134



```
reflect(reflect(x = 6)){
for (i < 3){
if (i > 0){
line(-2*i + 7,-4*i + 11,-2*i + 7
arrow = True,solid = True);
rectangle(0,-4*i + 11,6,-3*i + 1
}
rectangle(2,0,4,2)
}
}
```

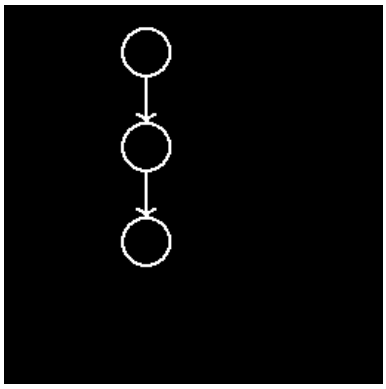
135

136



```
for (i < 3){
if (i > 0){
line(3*i,1,3*i + -1,1,
arrow = True,solid = True)
}
rectangle(3*i,0,3*i + 2,2)
}
```

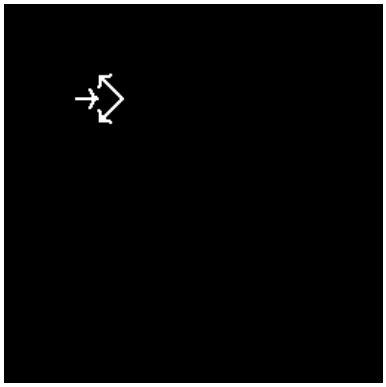
137



138

```
line(1,3,1,4,
arrow = False,solid = True);
for (i < 3){
if (i > 0){
line(1,-5*i + 13,1,-4*i + 10,
arrow = True,solid = True)
}
circle(1,-4*i + 9)
}
```

139

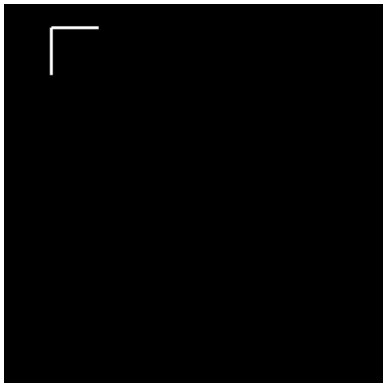


140

```
reflect(reflect(x = 2)){
line(0,1,1,2,
arrow = False,solid = True);
line(1,0,2,1,
arrow = False,solid = True)
}
```

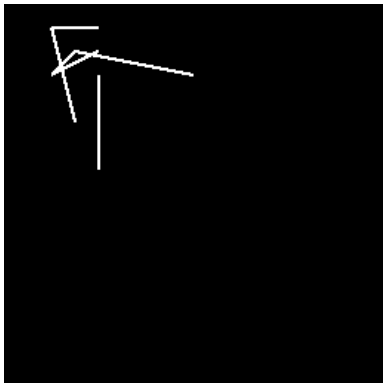
141

142



```
line(0,0,0,2,
arrow = False,solid = True);
line(0,2,2,2,
arrow = False,solid = True)
```

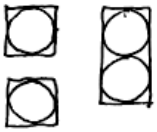
143

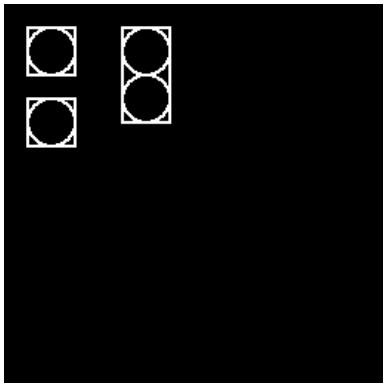


144

```
for (i < 3){
line(i,-1*i + 6,2*i + 2,-1*i + 6
arrow = False,solid = True);
line(i,-2*i + 4,i,-1*i + 6,
arrow = False,solid = True)
}
```

145



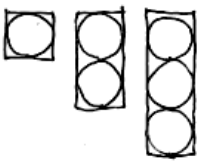


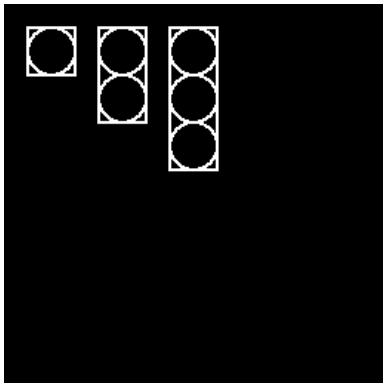
146

```
for (i < 3){
if (i > 0){
circle(1,-3*i + 7);
circle(5,-2*i + 6);
rectangle(0,-3*i + 6,2,-3*i + 8)
}
rectangle(4,1,6,5)
}
```

147

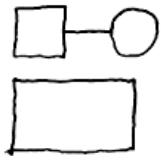
148



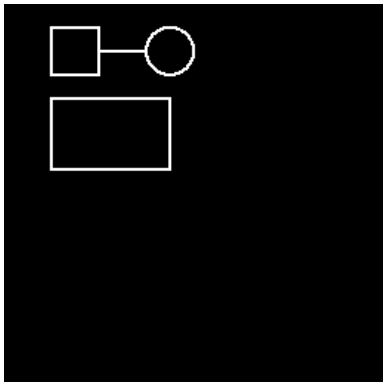


```
for (i < 3){
rectangle(3*i,-2*i + 4,3*i + 2,6
for (j < i + 1){
circle(3*i + 1,-2*j + 5)
}
}
```

149

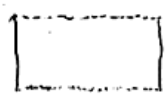


150

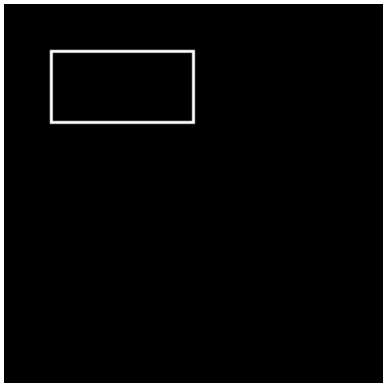


```
circle(5,5);
line(2,5,4,5,
arrow = False,solid = True);
rectangle(0,0,5,3);
rectangle(0,4,2,6)
```

151



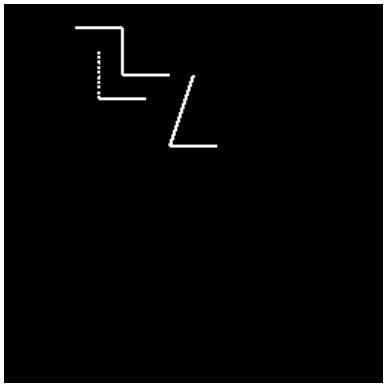
152



```
line(0,0,6,0,
arrow = False,solid = False);
reflect(reflect(x = 6)){
line(6,0,6,3,
arrow = False,solid = True);
line(0,3,6,3,
arrow = False,solid = False)
}
```

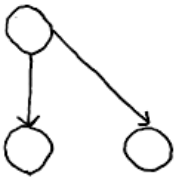
153

154

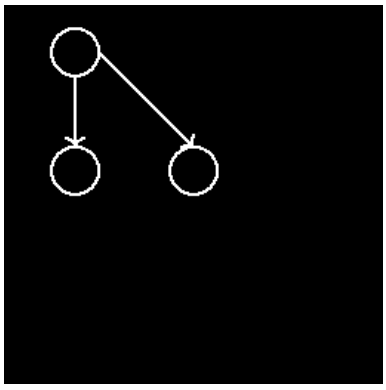


Solver timeout

155

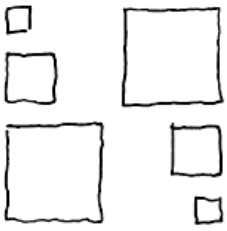


156

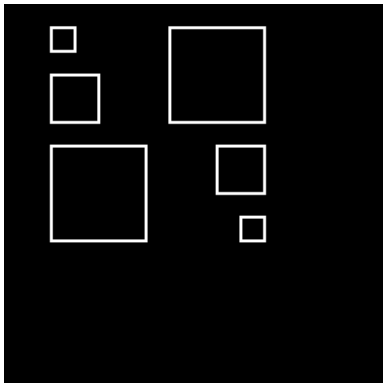


```
for (i < 3){
if (i > 0){
circle(-5*i + 11,1);
line(-1*i + 3,-1*i + 7,-5*i + 11
arrow = True,solid = True)
}
circle(1,6)
}
```

157



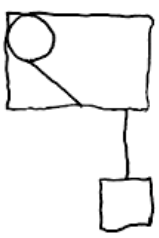
158

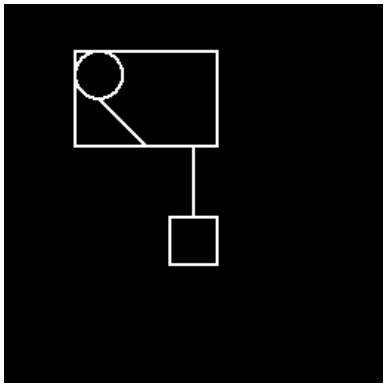


Solver timeout

159

160

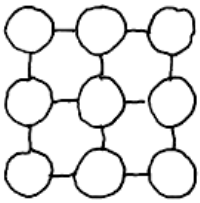


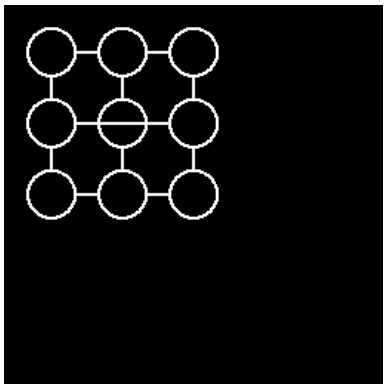


```
for (i < 3){
if (i > 0){
line(4*i + -3,-5*i + 12,2*i + 1,
arrow = False,solid = True);
rectangle(4*i + -4,-5*i + 10,6,-
}
circle(1,8)
}
```

161

162

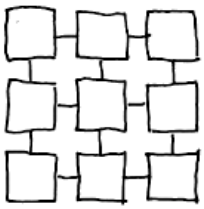


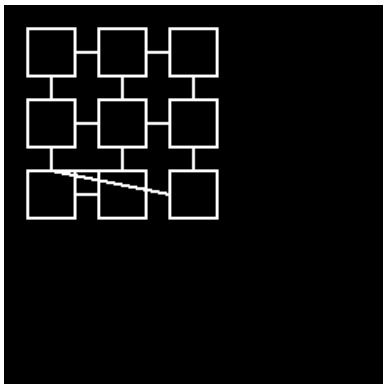


Solver timeout

163

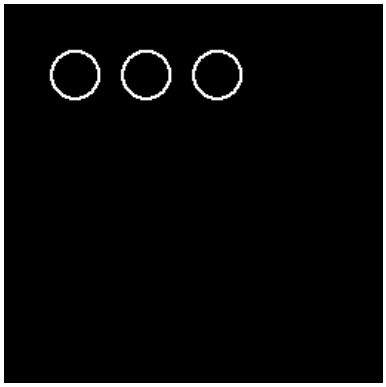
164





Solver timeout

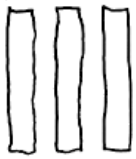
165

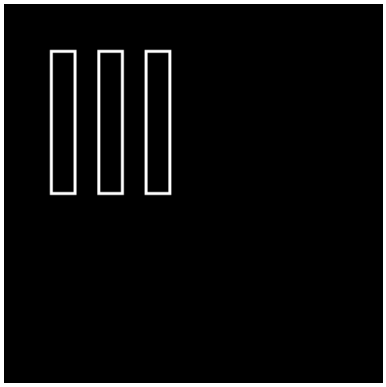


166

```
for (i < 3){
circle(-3*i + 7,1)
}
```

167

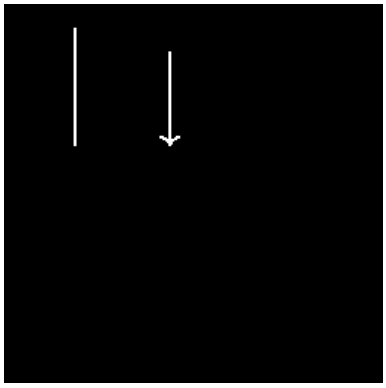




168

```
for (i < 3){
rectangle(-2*i + 4,0,-2*i + 5,6)
}
```

169

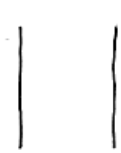


170

```
line(4,0,4,1,
arrow = False,solid = False);
line(0,0,0,5,
arrow = False,solid = False);
line(4,1,4,5,
arrow = False,solid = False)
```

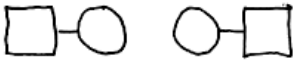
171

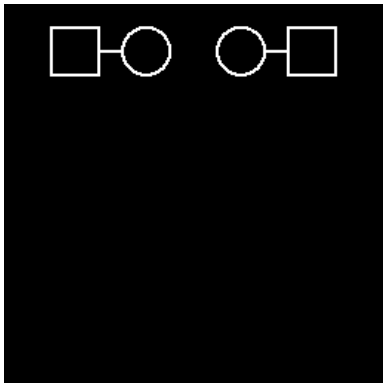
172



```
line(4,0,4,5,
arrow = False,solid = True);
line(0,0,0,5,
arrow = False,solid = True)
```

173

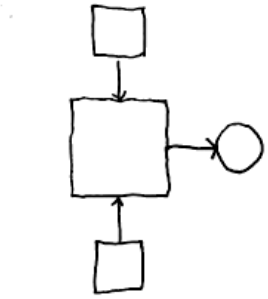


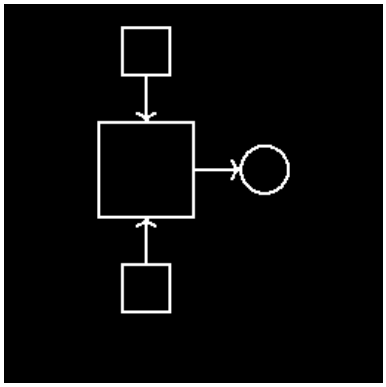


174

```
reflect(reflect(x = 12)){
circle(4,1);
line(9,1,10,1,
arrow = False,solid = True);
rectangle(0,0,2,2)
}
```

175



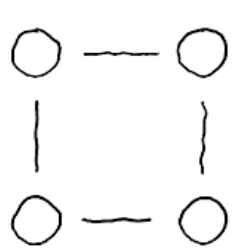


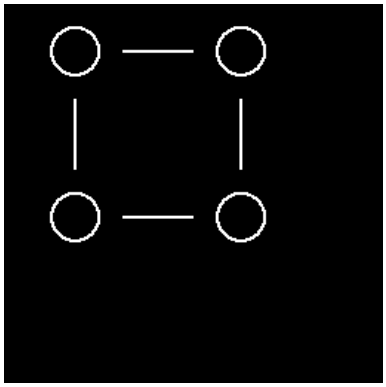
176

```
rectangle(0,4,4,8);
reflect(reflect(y = 12)){
circle(7,6);
line(2,2,2,4,
arrow = True,solid = True);
line(4,6,6,6,
arrow = True,solid = True);
rectangle(1,10,3,12)
}
```

177

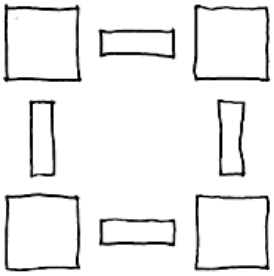
178

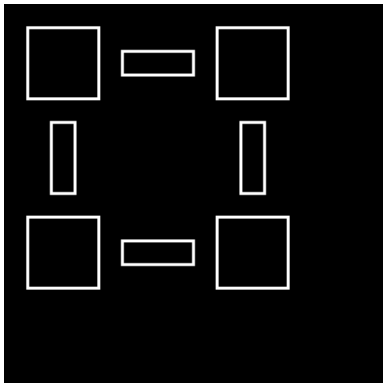




```
reflect(reflect(y = 9)){
line(3,8,6,8,
arrow = False,solid = True);
reflect(reflect(x = 9)){
circle(1,8);
line(1,3,1,6,
arrow = False,solid = True)
}
}
```

179





180

```
reflect(reflect(y = 11)){
rectangle(4,9,7,10);
reflect(reflect(x = 11)){
rectangle(1,4,2,7);
rectangle(8,8,11,11)
}
}
```

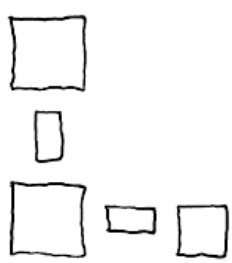
181

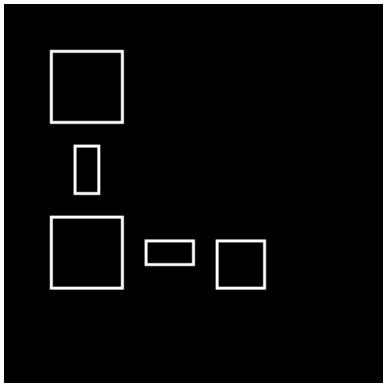
182

```
for (i < 4){
line(i,-1*i + 5,i + 2,-1*i + 5,
arrow = False,solid = True);
line(i + 2,-1*i + 3,i + 4,-1*i +
arrow = False,solid = True)
}
```

183

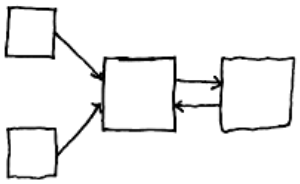
184



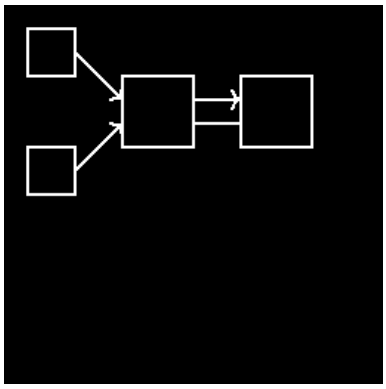


```
for (i < 3){
if (i > 0){
rectangle(3*i + 1,-1*i + 2,3*i +
rectangle(0,7*i + -7,3,7*i + -4)
}
rectangle(1,4,2,6)
}
```

185



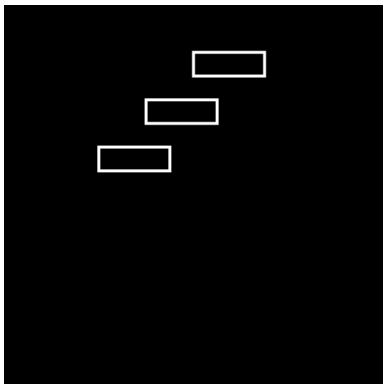
186



Solver timeout

187

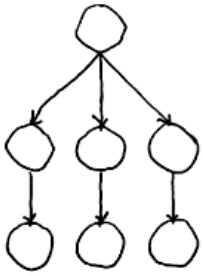
188

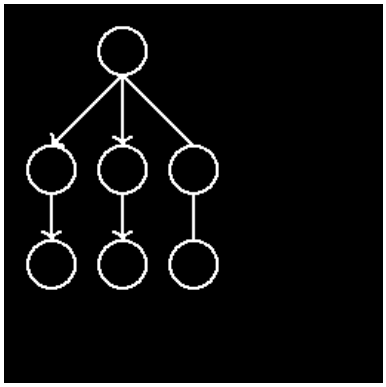


```
for (i < 3){
rectangle(-2*i + 4,-2*i + 4,-2*i
}
```

189

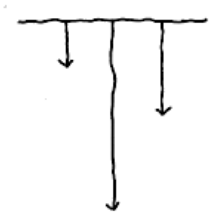
190



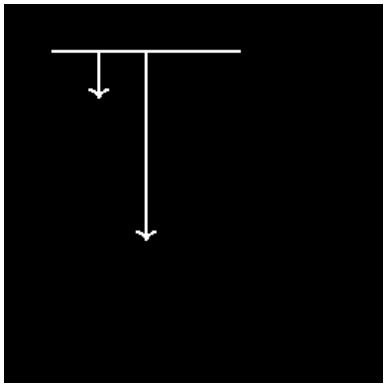


```
circle(4,10);
for (i < 3){
circle(-3*i + 7,5);
circle(-3*i + 7,1);
line(-3*i + 7,4,-3*i + 7,2,
arrow = True,solid = True);
line(4,9,-3*i + 7,6,
arrow = True,solid = True)
}
```

191

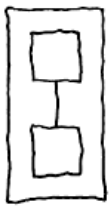


192

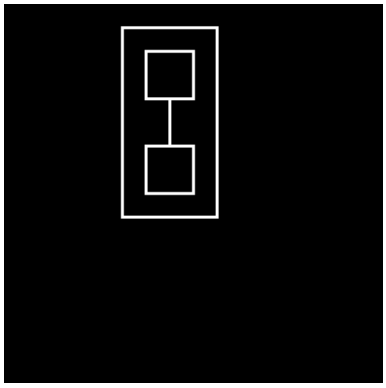


```
line(2,8,2,6,
arrow = True,solid = True);
line(6,8,6,4,
arrow = True,solid = True);
line(4,8,4,0,
arrow = True,solid = True);
line(0,8,8,8,
arrow = False,solid = True)
```

193



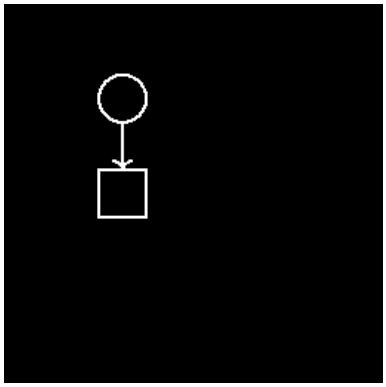
194



```
line(2,3,2,5,
arrow = False,solid = True);
rectangle(1,1,3,3);
rectangle(1,5,3,7);
rectangle(0,0,4,8)
```

195

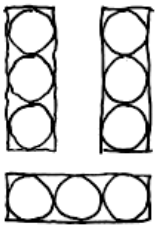
196

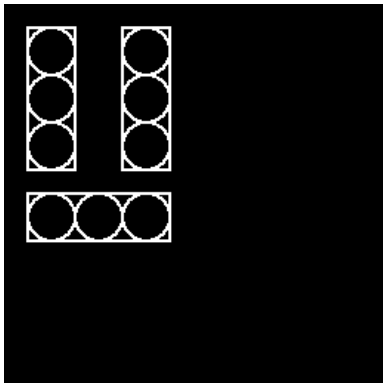


```
circle(1,5);
line(1,4,1,2,
arrow = True,solid = True);
rectangle(0,0,2,2)
```

197

198

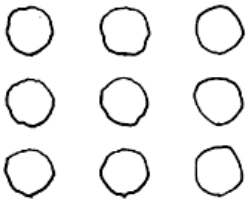


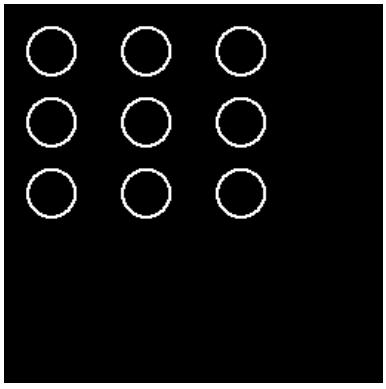


```
Solver timeout
```

199

200

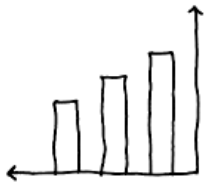


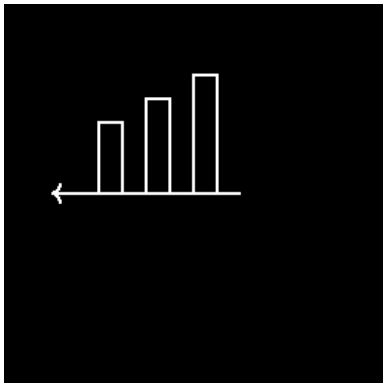


```
for (i < 3){
for (j < 3){
circle(-4*j + 9,-3*i + 7)
}
}
```

201

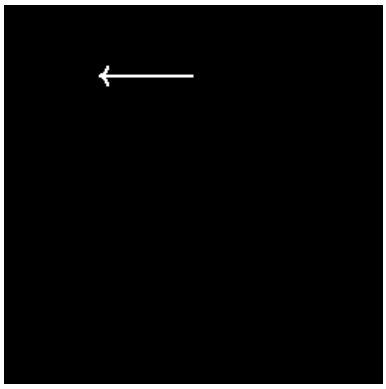
202





```
for (i < 3){
if (i > 0){
line(8,0,8*i + -8,7*i + -7,
arrow = True,solid = True)
}
rectangle(2*i + 2,0,2*i + 3,i +
}
```

203

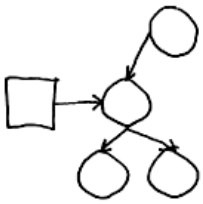


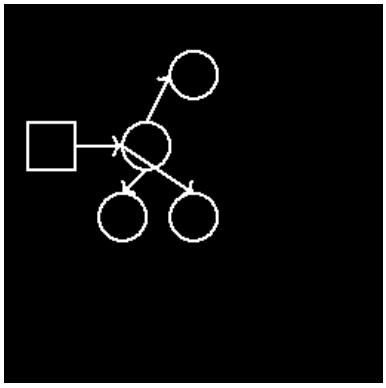
204

```
line(4,0,0,0,
arrow = False,solid = False)
```

205

206

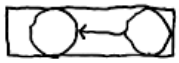


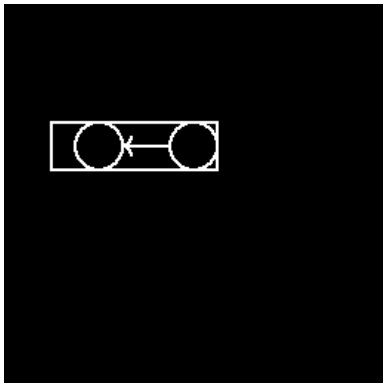


207

```
Solver timeout
```

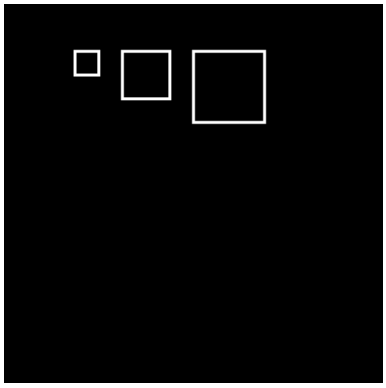
208





```
circle(2,1);
circle(6,1);
line(5,1,3,1,
arrow = True,solid = True);
rectangle(0,0,7,2)
```

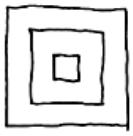
209

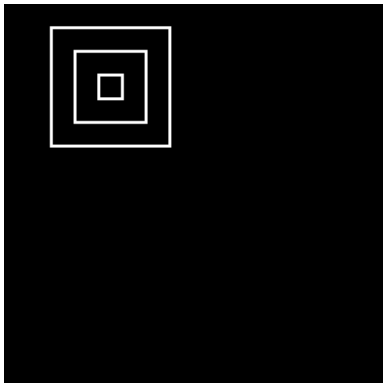


210

```
rectangle(5,0,8,3);
rectangle(2,1,4,3);
rectangle(0,2,1,3)
```

211



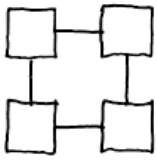


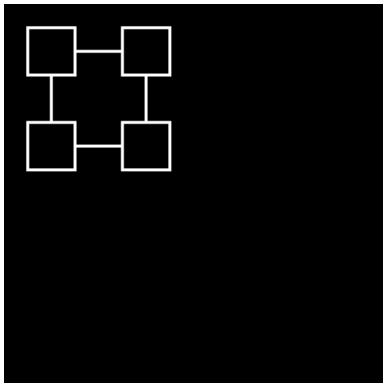
212

```
for (i < 3){
rectangle(-1*i + 2,-1*i + 2,i +
}
```

213

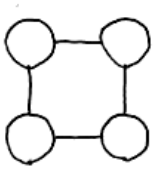
214

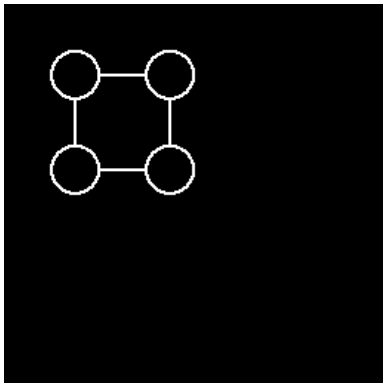




```
reflect(reflect(x = 6)){
line(5,2,5,4,
arrow = False,solid = True);
reflect(reflect(y = 6)){
line(2,1,4,1,
arrow = False,solid = True);
rectangle(4,4,6,6)
}
}
```

215

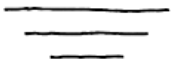


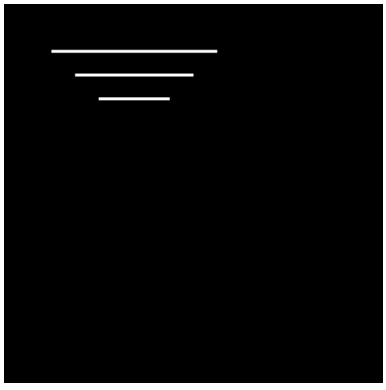


216

```
reflect(reflect(y = 6)){
line(2,5,4,5,
arrow = False,solid = True);
reflect(reflect(x = 6)){
circle(5,5);
line(1,2,1,4,
arrow = False,solid = True)
}
}
```

217



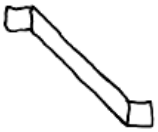


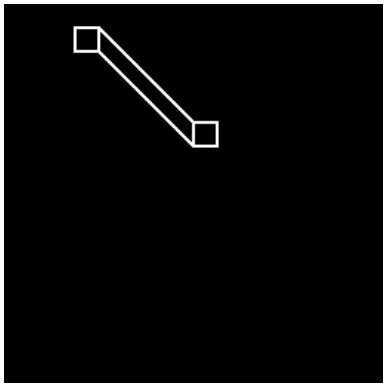
218

```
for (i < 3){
line(i,-1*i + 2,-1*i + 7,-1*i +
arrow = False,solid = True)
}
```

219

220

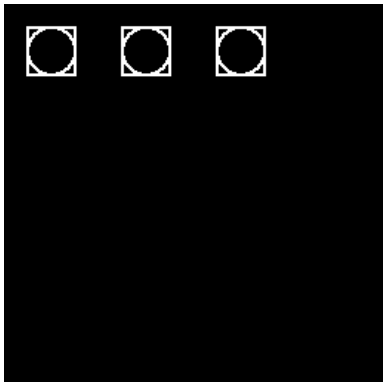




```
line(1,4,5,0,
arrow = False,solid = True);
line(1,5,5,1,
arrow = False,solid = True);
rectangle(5,0,6,1);
rectangle(0,4,1,5)
```

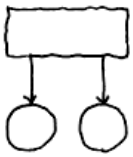
221

222

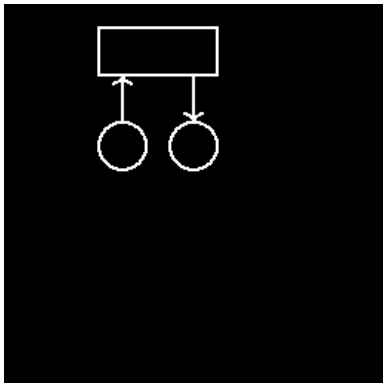


```
for (i < 3){
circle(4*i + 1,1);
rectangle(4*i,0,4*i + 2,2)
}
```

223

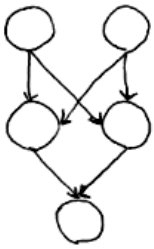


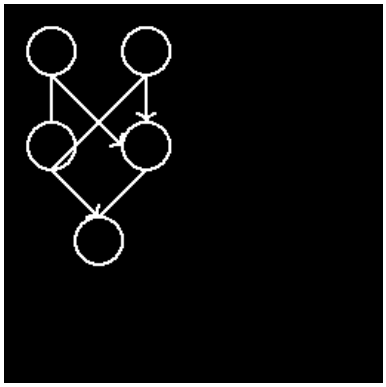
224



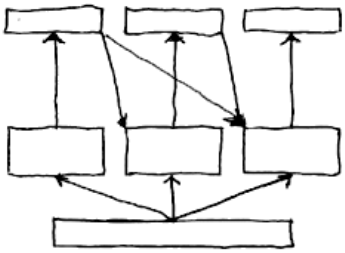
```
reflect(reflect(x = 5)){
circle(1,1);
line(4,4,4,2,
arrow = True,solid = True);
rectangle(0,4,5,6)
}
```

225



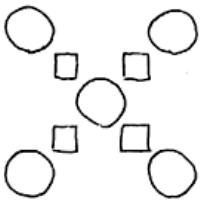


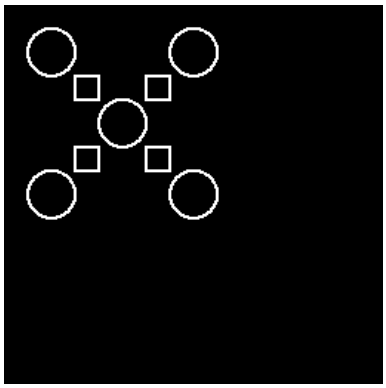
Solver timeout



Sampled no finished traces.

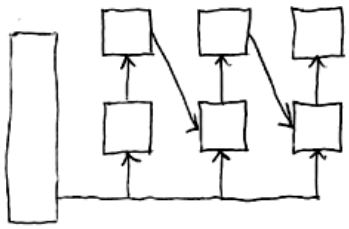
Solver timeout





```
reflect(reflect(y = 8)){
for (i < 3){
if (i > 0){
rectangle(3*i + -1,2,3*i,3)
}
circle(3*i + 1,3*i + 1)
}
}
```

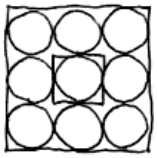
232



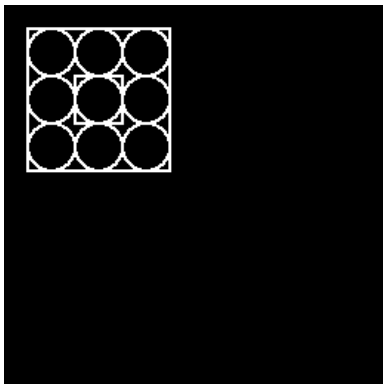
233

Sampled no finished traces.

```
Solver timeout
```

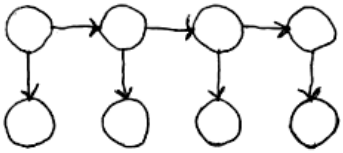


234

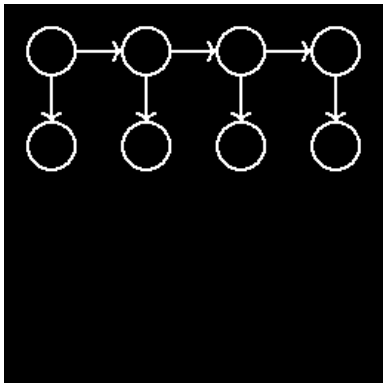


```
for (i < 3){
if (i > 0){
rectangle(-2*i + 4,-2*i + 4,2*i
}
for (j < 3){
circle(-2*i + 5,2*j + 1)
}
}
```

235



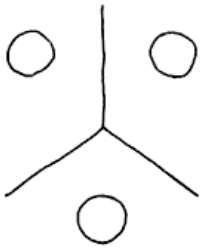
236

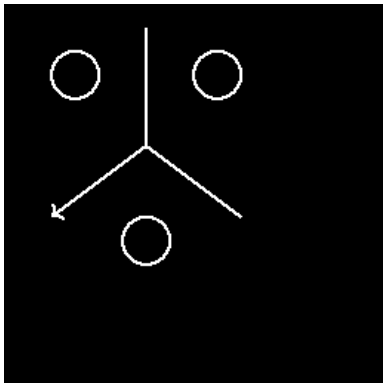


```
for (i < 4){
line(-4*i + 13,4,-4*i + 13,2,
arrow = True,solid = True);
for (j < 3){
if (j > 0){
circle(-4*i + 13,4*j + -3)
}
line(-4*j + 10,5,-4*j + 12,5,
arrow = True,solid = True)
}
}
```

237

238

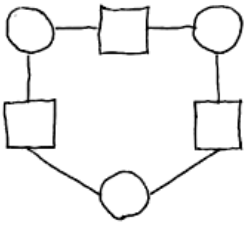


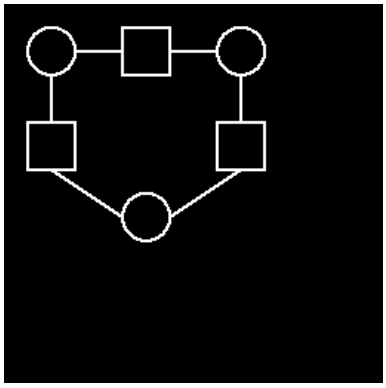


```
circle(4,1);
reflect(reflect(x = 8)){
circle(1,8);
line(4,5,8,2,
arrow = False,solid = True);
line(4,5,4,10,
arrow = False,solid = True)
}
```

239

240

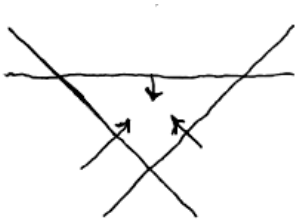


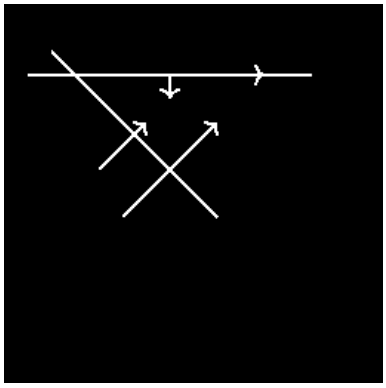


```
Solver timeout
```

241

242

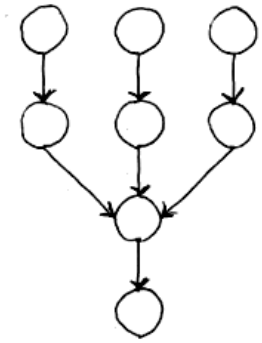


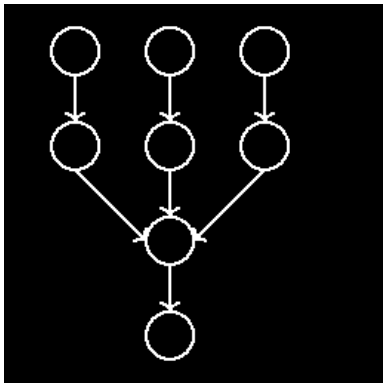


```
line(0,6,12,6,
arrow = False,solid = True);
line(6,6,6,5,
arrow = True,solid = True);
line(8,3,7,4,
arrow = True,solid = True);
line(3,2,5,4,
arrow = True,solid = True);
reflect(reflect(x = 12)){
line(4,0,12,8,
arrow = False,solid = True)
}
```

243

244

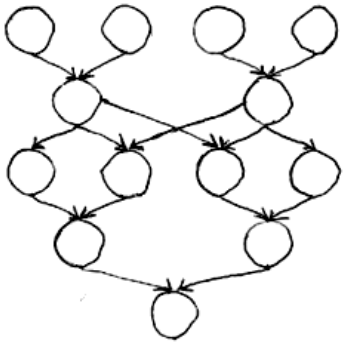


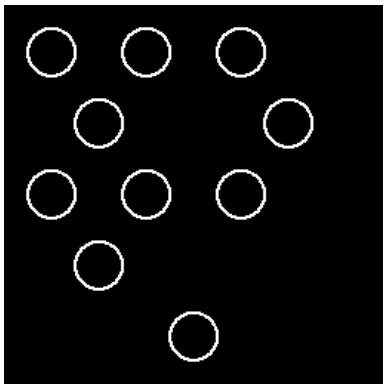


Solver timeout

245

246

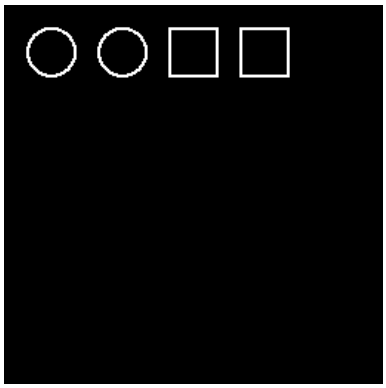




Solver timeout

247

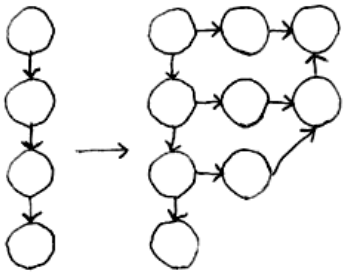
248

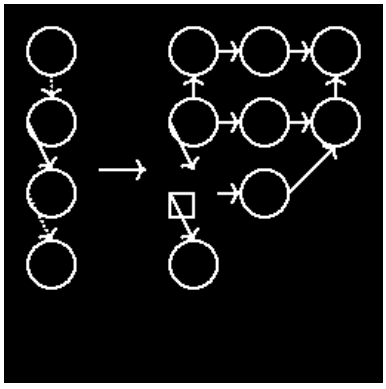


```
circle(1,1);
circle(4,1);
rectangle(9,0,11,2);
rectangle(6,0,8,2)
```

249

250

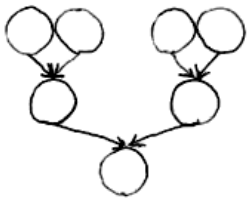


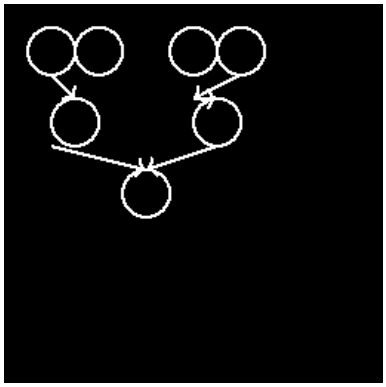


Solver timeout

251

252

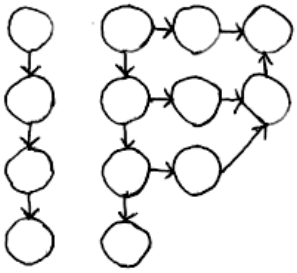


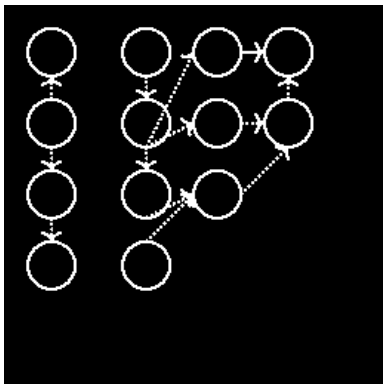


Solver timeout

253

254

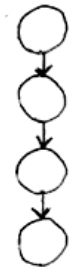


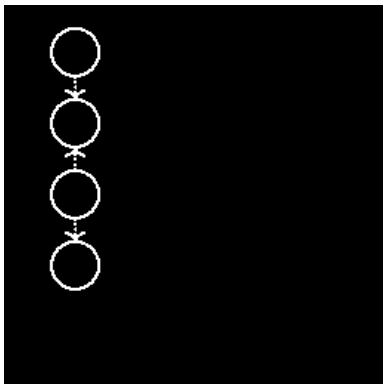


Solver timeout

255

256

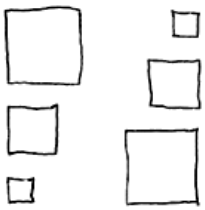


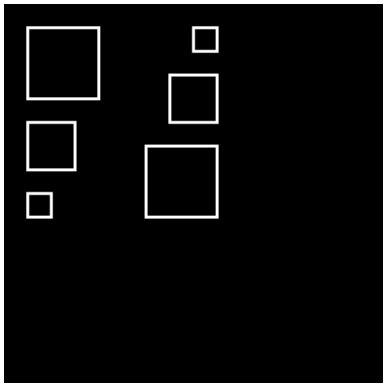


```
for (i < 4){
if (i > 0){
line(1,-3*i + 12,1,-3*i + 11,
arrow = True,solid = True)
}
circle(1,-3*i + 10)
}
```

257

258

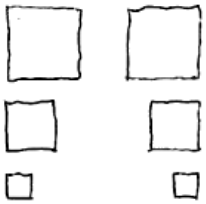


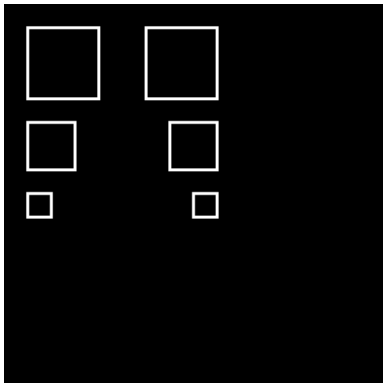


Solver timeout

259

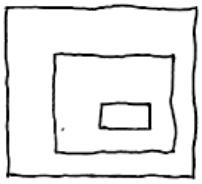
260



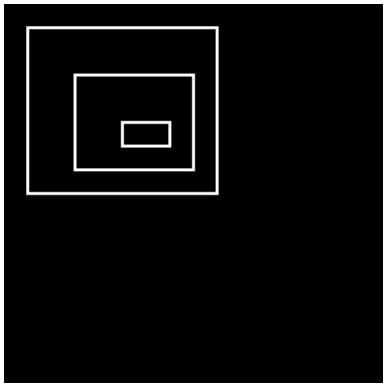


```
reflect(reflect(x = 8)){
rectangle(0,5,3,8);
rectangle(0,2,2,4);
rectangle(0,0,1,1)
}
```

261

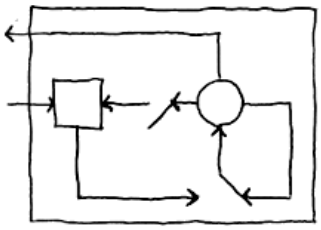


262

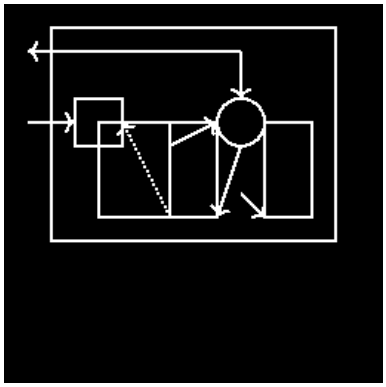


```
for (i < 3){
rectangle(-2*i + 4,-1*i + 2,i +
}
```

263



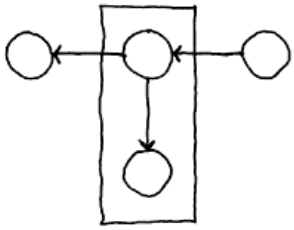
264

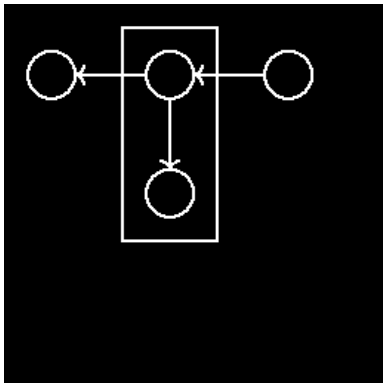


```
Solver timeout
```

265

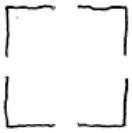
266

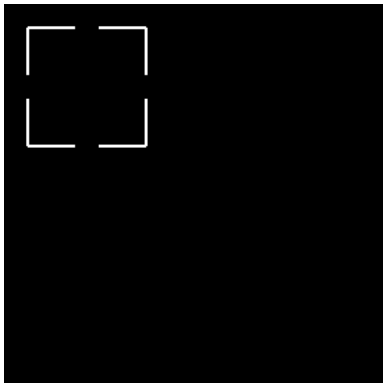




```
line(6,6,6,3,
arrow = True,solid = True);
for (i < 3){
if (i > 0){
circle(-5*i + 16,7);
circle(-5*i + 11,5*i + -3);
line(-5*i + 15,7,-5*i + 12,7,
arrow = True,solid = True)
}
rectangle(4,0,8,9)
}
```

267

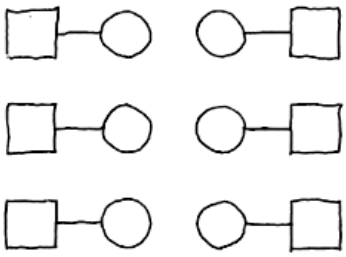


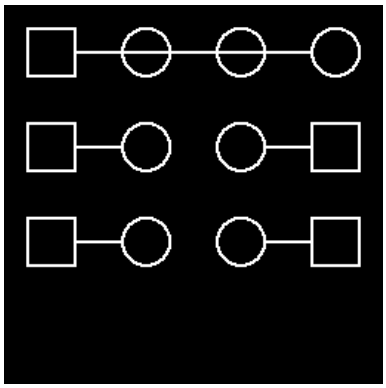


268

```
reflect(reflect(x = 5)){
reflect(reflect(y = 5)){
line(5,3,5,5,
arrow = False,solid = True);
line(3,5,5,5,
arrow = False,solid = True)
}
}
```

269



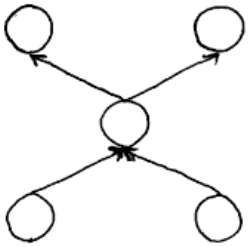


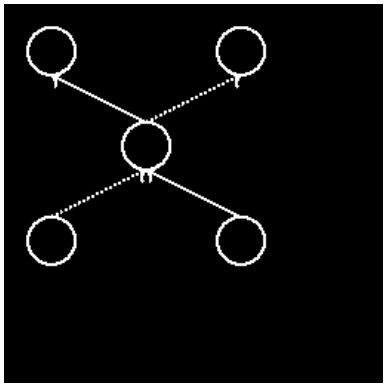
270

```
reflect(reflect(x = 14)){
for (i < 3){
circle(9,-4*i + 9);
line(10,-4*i + 9,12,-4*i + 9,
arrow = False,solid = True);
rectangle(0,-4*i + 8,2,-4*i + 10
}
}
```

271

272

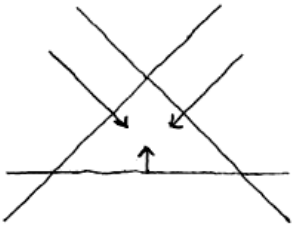




```
reflect(reflect(x = 10)){
for (i < 3){
if (i > 0){
line(4*i + -3,4*i + -2,4*i + 1,4
arrow = True,solid = True)
}
circle(4*i + 1,4*i + 1)
}
}
```

273

274

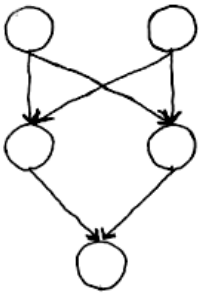


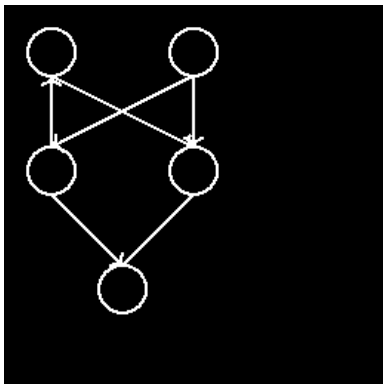
Sampled no finished traces.

```
line(0,2,12,2,
arrow = False,solid = True);
line(6,2,6,3,
arrow = True,solid = True);
reflect(reflect(x = 12)){
line(0,0,9,9,
arrow = False,solid = True);
line(10,7,7,4,
arrow = True,solid = True)
}
```

275

276

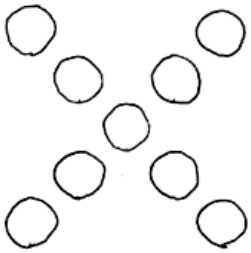


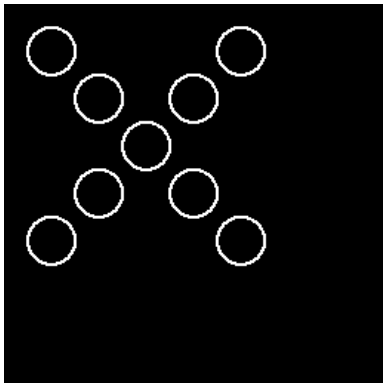


```
reflect(reflect(x = 8)){
circle(4,1);
for (i < 3){
if (i > 0){
circle(7,-5*i + 16);
line(-6*i + 13,10,7,7,
arrow = True,solid = True)
}
line(1,5,4,2,
arrow = True,solid = True)
}
}
```

277

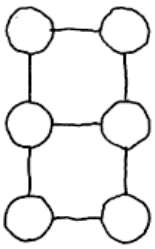
278

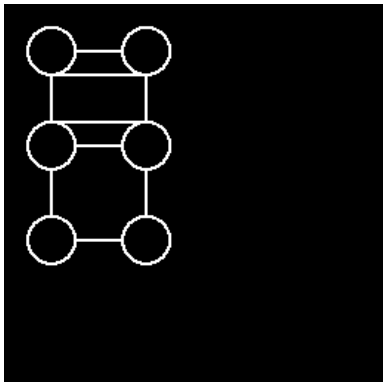




```
reflect(reflect(x = 10)){
circle(1,1);
for (i < 4){
circle(-2*i + 7,2*i + 3)
}
}
```

279

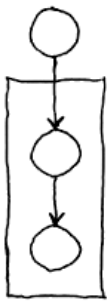


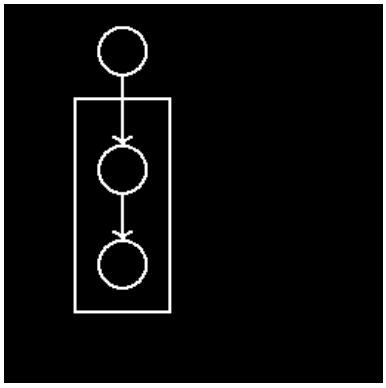


280

```
reflect(reflect(x = 6)){
for (i < 3){
if (i > 0){
line(1,-4*i + 10,1,-4*i + 12,
arrow = False,solid = True)
}
circle(5,-4*i + 9);
line(2,-4*i + 9,4,-4*i + 9,
arrow = False,solid = True)
}
}
```

281



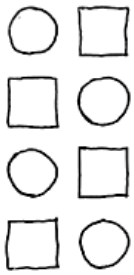


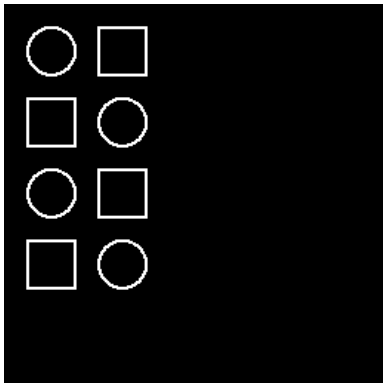
282

```
rectangle(0,0,4,9);
for (i < 3){
if (i > 0){
circle(2,-4*i + 10);
line(2,-5*i + 15,2,-4*i + 11,
arrow = True,solid = True)
}
circle(2,11)
}
```

283

284





```
for (i < 2){
circle(4,-6*i + 7);
circle(1,-6*i + 10);
rectangle(0,-6*i + 6,2,-6*i + 8)
rectangle(3,-6*i + 9,5,-6*i + 11
}
```