Supplement to: Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis
MIT
ellisk@mit.edu

Daniel Ritchie
Stanford University
dritchie@stanford.edu

Armando Solar-Lezama
MIT
asolar@csail.mit.edu

Joshua B. Tenenbaum
MIT
jbt@mit.edu

1 Correcting errors made by the neural network

The program synthesizer can help correct errors from the execution trace proposal network by favoring execution traces which lead to more concise or general programs. For example, one generally prefers figures with perfectly aligned objects over figures whose parts are slightly misaligned – and precise alignment lends itself to short programs. Similarly, figures often have repeated parts, which the program synthesizer might be able to model as a loop or reflectional symmetry. So, in considering several candidate traces proposed by the neural network, we might prefer traces whose best programs have desirable features such being short or having iterated structures.

Concretely, we implemented the following scheme: for an image I, the neurally guided sampling scheme of section 3 of the main paper samples a set of candidate traces, written $\mathcal{F}(I)$. Instead of predicting the most likely trace in $\mathcal{F}(I)$ according to the neural network, we can take into account the programs that best explain the traces. Writing $\hat{T}(I)$ for the trace the model predicts for image I,

$$\hat{T}(I) = \underset{T \in \mathcal{F}(I)}{\arg\max}\, L_{\text{learned}}(I|\text{render}(T)) \times \mathbb{P}_\theta[T|I] \times \mathbb{P}_\beta[\text{program}(T)] \tag{1}$$

where $\mathbb{P}_\beta[\cdot]$ is a prior probability distribution over programs parameterized by β. This is equivalent to doing MAP inference in a generative model where the program is first drawn from $\mathbb{P}_\beta[\cdot]$, then the program is executed deterministically, and then we observe a noisy version of the program's output, where $L_{\text{learned}}(I|\text{render}(\cdot)) \times \mathbb{P}_\theta[\cdot|I]$ is our observation model.

Given a corpus of graphics program synthesis problems with annotated ground truth traces (i.e. (I, T) pairs), we find a maximum likelihood estimate of β:

$$\beta^* = \arg\max_\beta \left[\log \frac{\mathbb{P}_\beta[\text{program}(T)] \times L_{\text{learned}}(I|\text{render}(T)) \times \mathbb{P}_\theta[T|I]}{\sum_{T' \in \mathcal{F}(I)} \mathbb{P}_\beta[\text{program}(T')] \times L_{\text{learned}}(I|\text{render}(T')) \times \mathbb{P}_\theta[T'|I]}\right] \tag{2}$$

where the expectation is taken both over the model predictions and the (I, T) pairs in the training corpus. We define $\mathbb{P}_\beta[\cdot]$ to be a log linear distribution $\propto \exp(\beta \cdot \phi(\text{program}))$, where $\phi(\cdot)$ is a feature extractor for programs. We extract a few basic features of a program, such as its size and how many loops it has, and use these features to help predict whether a trace is the correct explanation for an image.

We synthesized programs for the top 10 traces output by the deep network. Learning this prior over programs can help correct mistakes made by the neural network, and also occasionally introduces mistakes of its own; see Fig. 1 for a representative example of the kinds of corrections that it makes. On the whole it modestly improves our Top-1 accuracy from 58% to 61%. Recall that from Fig. 6 of the main paper that the best improvement in accuracy we could possibly get is 65% by looking at the top 10 traces.

2 Neural network architecture

2.1 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.

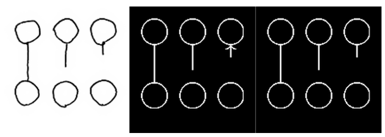


Figure 1: Left: hand drawing. Center: interpretation favored by the deep network. Right: interpretation favored after learning a prior over programs. Our learned prior favors shorter, simpler programs, thus continuing the pattern of not having an arrow is preferred.

2.2 Autoregressive decoding of drawing commands

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

$$\mathbb{P}[T_1] \propto W_{T_1} f \tag{3}$$

where W_{T_1} is a learned weight matrix.

Subsequent tokens are predicted as:

$$\mathbb{P}[T_n | T_{1:(n-1)}] \propto \text{MLP}_{T_1,n}(I \otimes \bigotimes_{j<n} \text{oneHot}(T_j)) \tag{4}$$

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).

2.3 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.

3 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

4 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:

$$-\log L(I_1|I_2) = \sum_{1\leq x\leq 256} \sum_{1\leq y\leq 256} |I_1[x,y] - I_2[x,y]| \begin{cases} \alpha, \text{if } \mathrm{I}_1[x,y] > I_2[x,y] \\ \beta, \text{if } \mathrm{I}_1[x,y] < I_2[x,y] \\ 0, \text{if } \mathrm{I}_1[x,y] = I_2[x,y] \end{cases} \tag{5}$$

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 $\beta = 0.04$.

5 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 [1] 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 2 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`.

6 The cost function for programs

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

$$\text{program}(T) = \underset{\substack{p\in\text{DSL} \\ p \text{ evaluates to } T}}{\arg\min} \; \text{cost}(p) \tag{6}$$

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

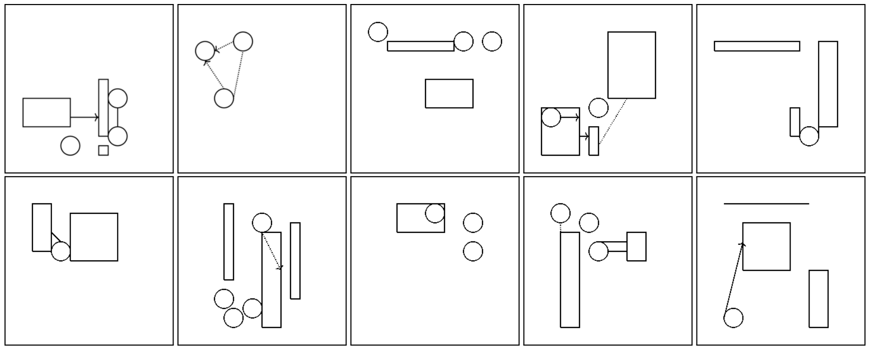


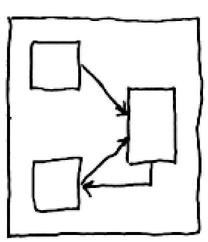
Figure 2: Example synthetic training data

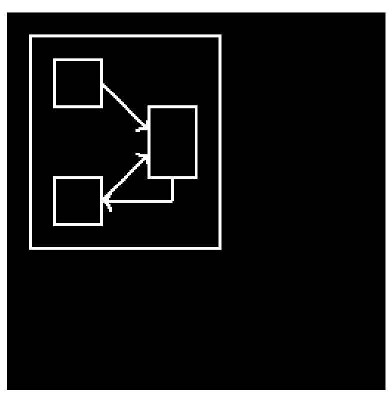
is effectively a prior over the space of program so that it tends to produce programs that have this symmetry.

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.

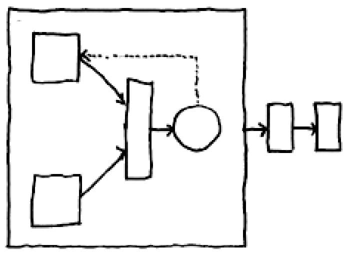
7 Full results on drawings data set

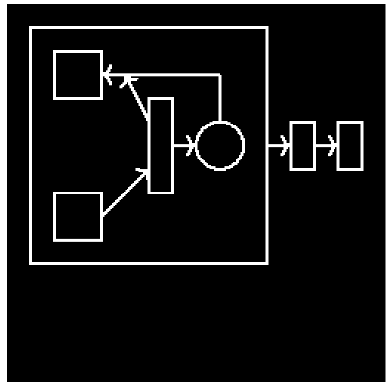
Below we show our full data set of drawings. The leftmost column is a hand drawing. The middle column is a rendering of the most likely trace discovered by the neurally guided SMC sampling scheme. The rightmost column is the program we synthesized from a ground truth execution trace of the drawing.



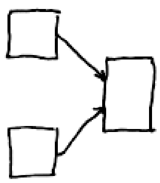


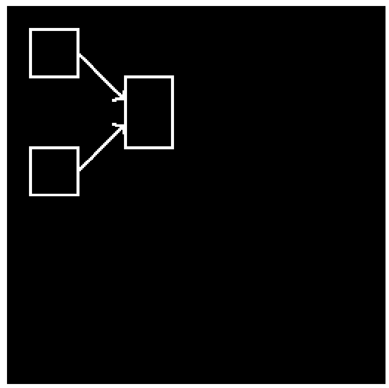
```
line(6,2,6,3,
arrow = False,solid = True);
line(6,2,3,2,
arrow = True,solid = True);
reflect(reflect(y = 9)){
line(3,2,5,4,
arrow = True,solid = True);
rectangle(0,0,8,9);
rectangle(5,3,7,6);
rectangle(1,1,3,3)
}
```



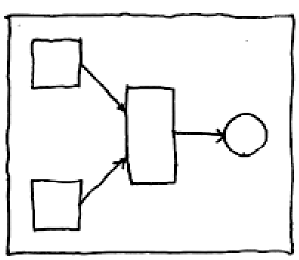


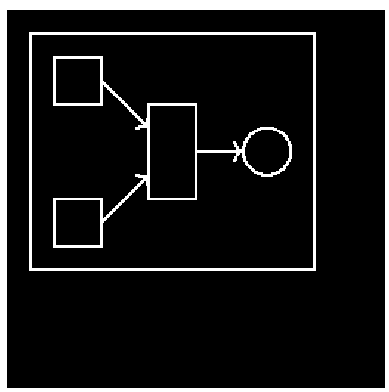
Solver timeout



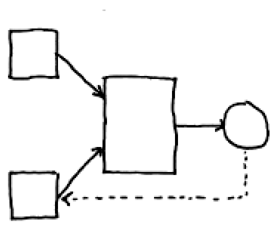


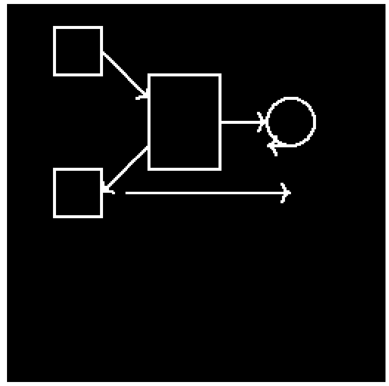
```
rectangle(4,2,6,5);
reflect(reflect(y = 7)){
line(2,6,4,4,
arrow = True,solid = True);
rectangle(0,0,2,2)
}
```



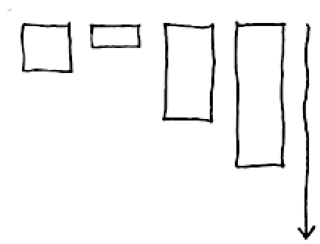


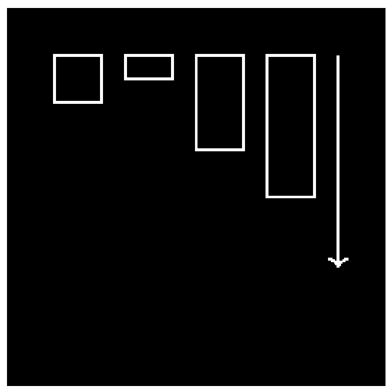
```
circle(10,5);
line(7,5,9,5,
arrow = True,solid = True);
rectangle(5,3,7,7);
rectangle(0,0,12,10);
reflect(reflect(y = 10)){
line(3,8,5,6,
arrow = True,solid = True);
rectangle(1,7,3,9)
}
```



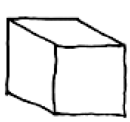


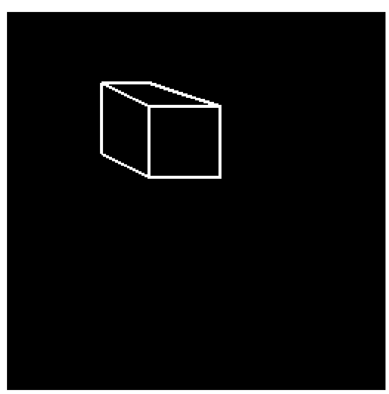
```
circle(10,4);
line(10,1,2,1,
arrow = True,solid = False);
line(10,1,10,3,
arrow = False,solid = False);
line(7,4,9,4,
arrow = True,solid = True);
reflect(reflect(y = 8)){
line(2,7,4,5,
arrow = True,solid = True);
rectangle(4,2,7,6);
rectangle(0,6,2,8)
}
```



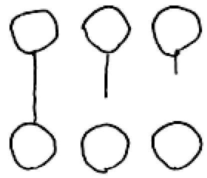
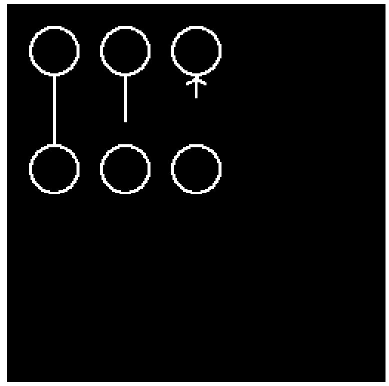


```
for (i < 3){
if (i > 0){
rectangle(-6*i + 12,2*i + 3,-6*i
rectangle(-6*i + 15,5*i + -2,-6*
}
line(12,9,12,0,
arrow = True,solid = True)
}
```



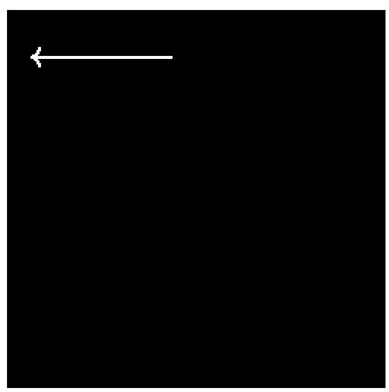


```
line(0,1,2,0,
arrow = False,solid = True);
for (i < 3){
if (i > 0){
line(3*i + -3,4,3*i + -1,3,
arrow = False,solid = True);
line(0,3*i + -2,3*i + -3,4,
arrow = False,solid = True)
}
rectangle(2,0,5,3)
}
```

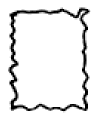



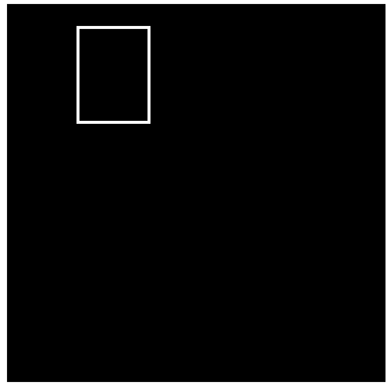
```
for (i < 3){
circle(-3*i + 7,1);
circle(-3*i + 7,6);
line(-3*i + 7,-1*i + 4,-3*i + 7,
arrow = False,solid = True)
}
```


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

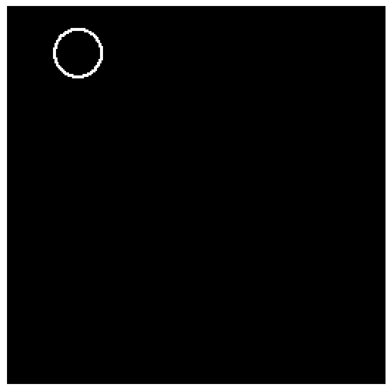



```
line(6,0,0,0,
arrow = True,solid = True)
```

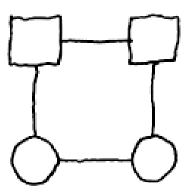


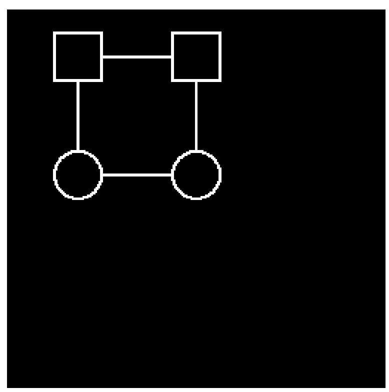


```
rectangle(0,0,3,4)
```

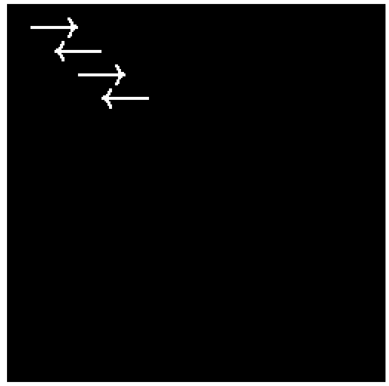


```
circle(1,1)
```

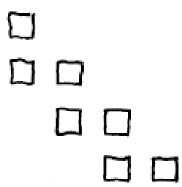


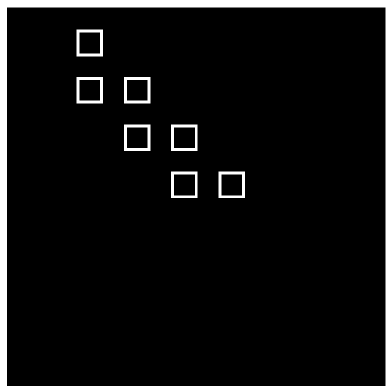


```
line(2,6,5,6,
arrow = False,solid = True);
reflect(reflect(x = 7)){
circle(6,1);
line(2,1,5,1,
arrow = False,solid = True);
line(1,2,1,5,
arrow = False,solid = True);
rectangle(5,5,7,7)
}
```

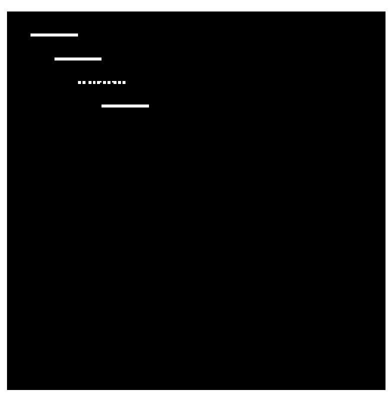


```
line(2,1,4,1,
arrow = True,solid = True);
line(3,2,1,2,
arrow = True,solid = True);
line(5,0,3,0,
arrow = True,solid = True);
line(0,3,2,3,
arrow = True,solid = True)
```

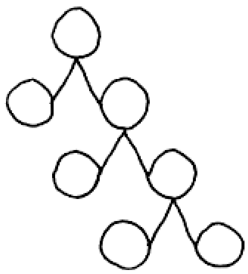


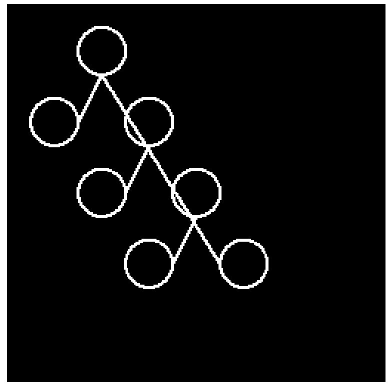


```
for (i < 4){
if (i > 0){
rectangle(-2*i + 6,2*i + -2,-2*i
}
rectangle(-2*i + 6,2*i,-2*i + 7,
}
```

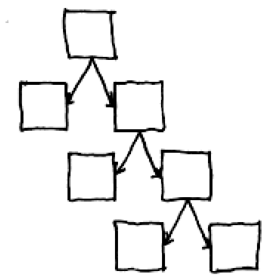


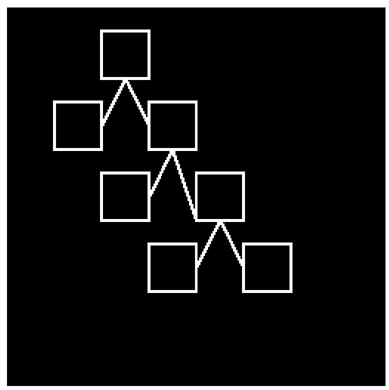
```
line(0,3,2,3,
arrow = False,solid = False);
line(2,1,4,1,
arrow = False,solid = False);
line(1,2,3,2,
arrow = False,solid = True);
line(3,0,5,0,
arrow = False,solid = True)
```



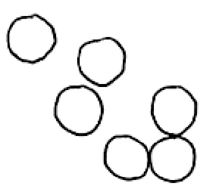


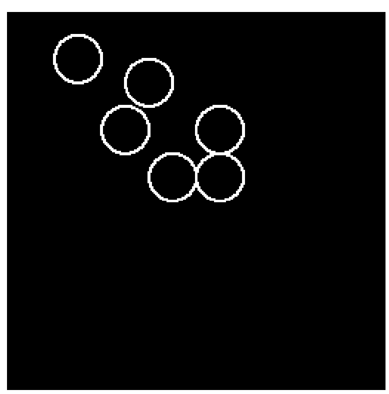
```
for (i < 4){
if (i > 0){
circle(-2*i + 7,3*i + -2);
line(-2*i + 9,3*i,-2*i + 10,3*i
arrow = False,solid = True);
line(-2*i + 8,3*i + -2,-2*i + 9,
arrow = False,solid = True)
}
circle(-2*i + 9,3*i + 1)
}
```



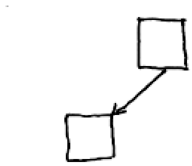


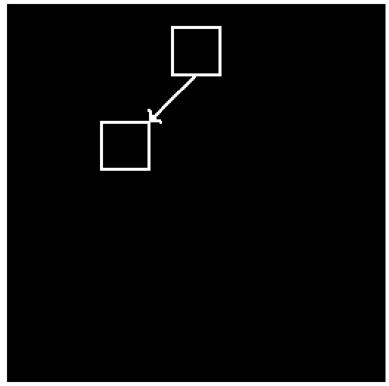
```
for (i < 4){
if (i > 0){
line(2*i + 1,-3*i + 12,2*i,-3*i
arrow = True,solid = True);
line(2*i + 1,-3*i + 12,2*i + 2,-
arrow = True,solid = True);
rectangle(2*i + -2,-3*i + 9,2*i,
}
rectangle(2*i + 2,-3*i + 9,2*i +
}
```



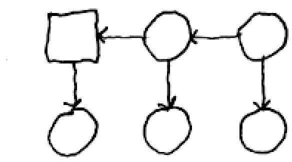


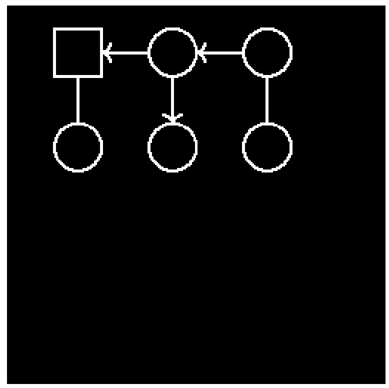
```
circle(5,1);
for (i < 3){
if (i > 0){
circle(7,2*i + -1);
circle(i + 2,2*i + 1)
}
circle(1,6)
}
```



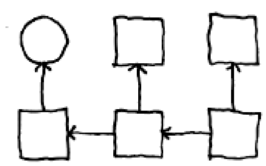


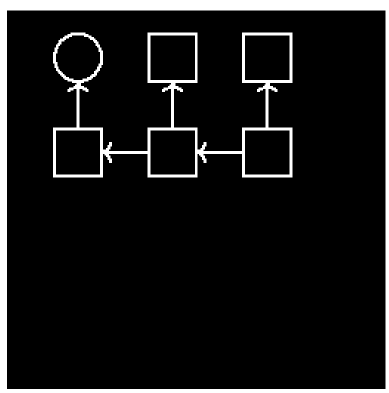
```
line(4,4,2,2,
arrow = True,solid = True);
rectangle(3,4,5,6);
rectangle(0,0,2,2)
```



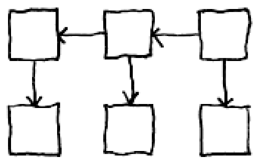


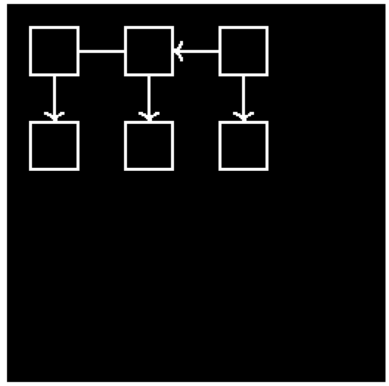
```
rectangle(0,4,2,6);
for (i < 3){
if (i > 0){
line(-4*i + 12,5,-4*i + 10,5,
arrow = True,solid = True);
for (j < i + 1){
circle(-4*j + 9,-4*i + 9)
}
}
line(-4*i + 9,4,-4*i + 9,2,
arrow = True,solid = True)
}
```



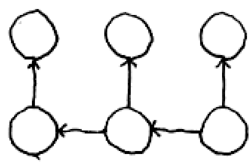


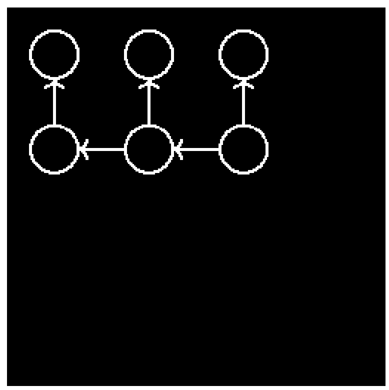
```
circle(1,5);
for (i < 3){
if (i > 0){
line(-4*i + 12,1,-4*i + 10,1,
arrow = True,solid = True);
rectangle(-4*i + 12,4,-4*i + 14,
}
line(-4*i + 9,2,-4*i + 9,4,
arrow = True,solid = True);
rectangle(-4*i + 8,0,-4*i + 10,2
}
```



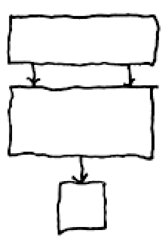


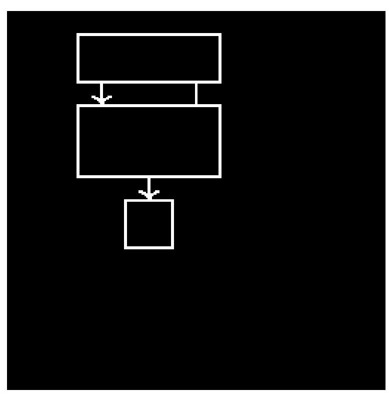
```
for (i < 3){
line(-4*i + 9,4,-4*i + 9,2,
arrow = True,solid = True);
for (j < 2){
line(-4*j + 8,5,-4*j + 6,5,
arrow = True,solid = True);
rectangle(-4*i + 8,4*j,-4*i + 10
}
}
```



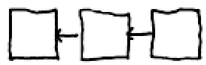


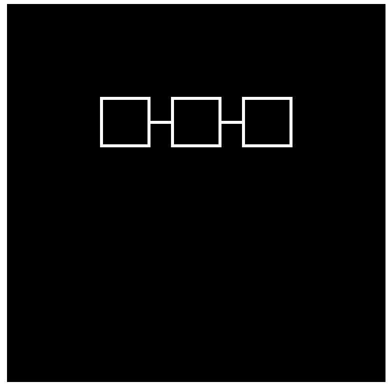
```
for (i < 3){
if (i > 0){
line(-4*i + 12,1,-4*i + 10,1,
arrow = True,solid = True)
}
circle(-4*i + 9,1);
circle(-4*i + 9,5);
line(-4*i + 9,2,-4*i + 9,4,
arrow = True,solid = True)
}
```



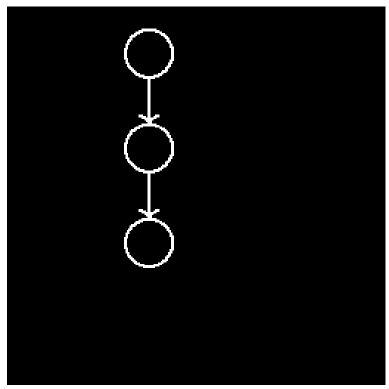


```
reflect(reflect(x = 6)){
for (i < 3){
if (i > 0){
line(-2*i + 7,-4*i + 11,-2*i + 7
arrow = True,solid = True);
rectangle(0,-4*i + 11,6,-3*i + 1
}
rectangle(2,0,4,2)
}
}
```

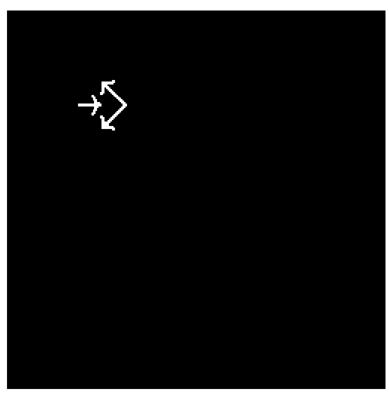




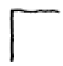
```
for (i < 3){
if (i > 0){
line(3*i,1,3*i + -1,1,
arrow = True,solid = True)
}
rectangle(3*i,0,3*i + 2,2)
}
```

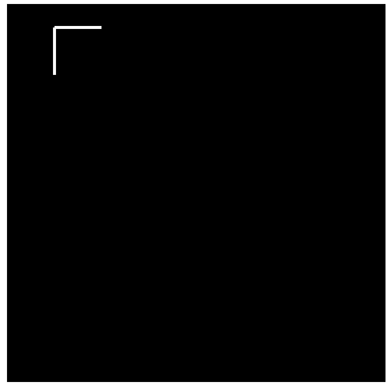


```
line(1,3,1,4,
arrow = False,solid = True);
for (i < 3){
if (i > 0){
line(1,-5*i + 13,1,-4*i + 10,
arrow = True,solid = True)
}
circle(1,-4*i + 9)
}
```

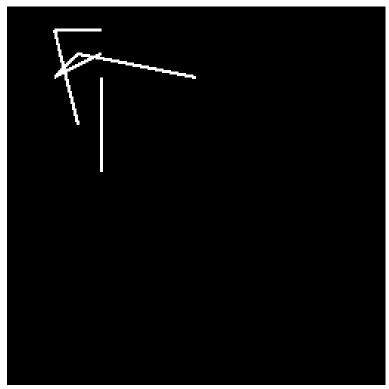


```
reflect(reflect(x = 2)){
line(0,1,1,2,
arrow = False,solid = True);
line(1,0,2,1,
arrow = False,solid = True)
}
```

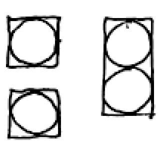


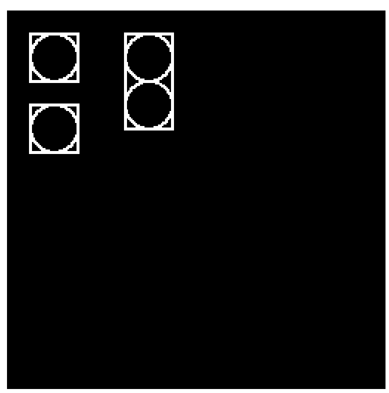


```
line(0,0,0,2,
arrow = False,solid = True);
line(0,2,2,2,
arrow = False,solid = True)
```

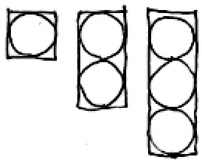


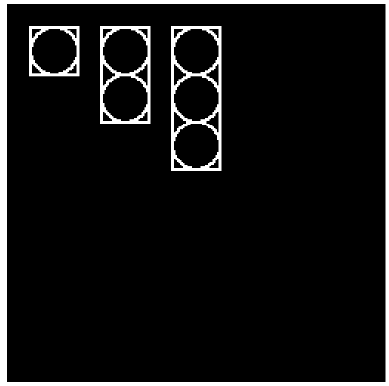
```
for (i < 3){
line(i,-1*i + 6,2*i + 2,-1*i + 6
arrow = False,solid = True);
line(i,-2*i + 4,i,-1*i + 6,
arrow = False,solid = True)
}
```



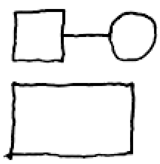


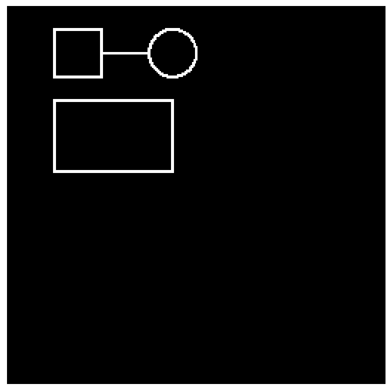
```
for (i < 3){
if (i > 0){
circle(1,-3*i + 7);
circle(5,-2*i + 6);
rectangle(0,-3*i + 6,2,-3*i + 8)
}
rectangle(4,1,6,5)
}
```



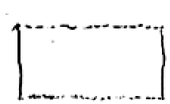


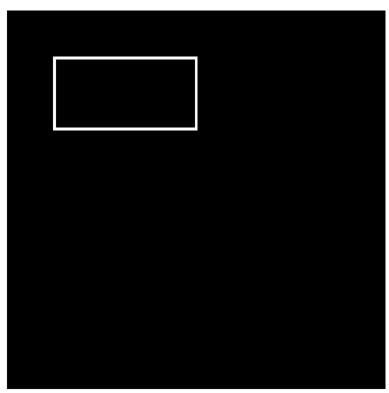
```
for (i < 3){
rectangle(3*i,-2*i + 4,3*i + 2,6
for (j < i + 1){
circle(3*i + 1,-2*j + 5)
}
}
```



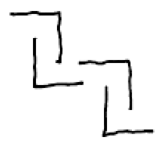


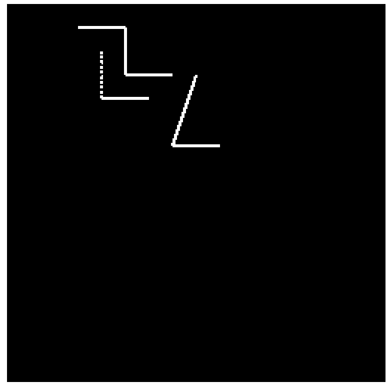
```
circle(5,5);
line(2,5,4,5,
arrow = False,solid = True);
rectangle(0,0,5,3);
rectangle(0,4,2,6)
```



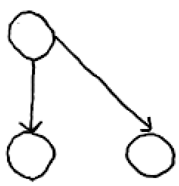


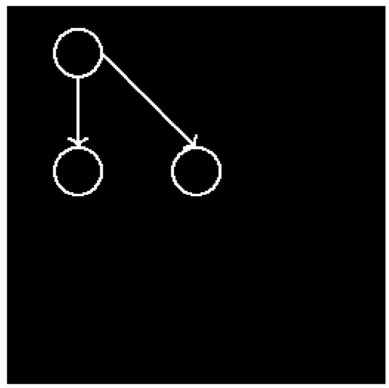
```
line(0,0,6,0,
arrow = False,solid = False);
reflect(reflect(x = 6)){
line(6,0,6,3,
arrow = False,solid = True);
line(0,3,6,3,
arrow = False,solid = False)
}
```



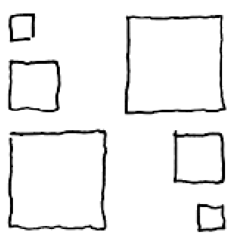


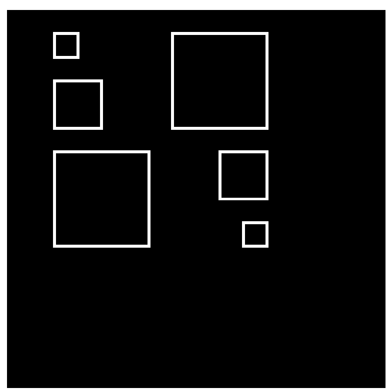
Solver timeout



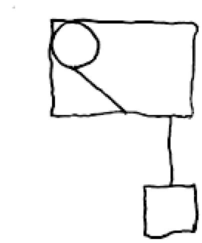


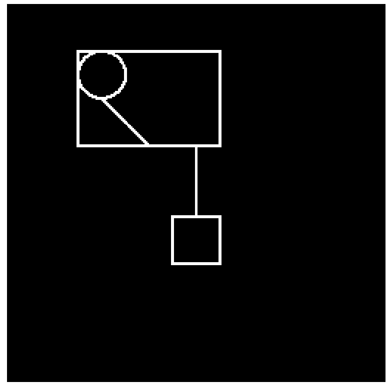
```
for (i < 3){
if (i > 0){
circle(-5*i + 11,1);
line(-1*i + 3,-1*i + 7,-5*i + 11
arrow = True,solid = True)
}
circle(1,6)
}
```



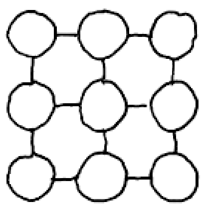


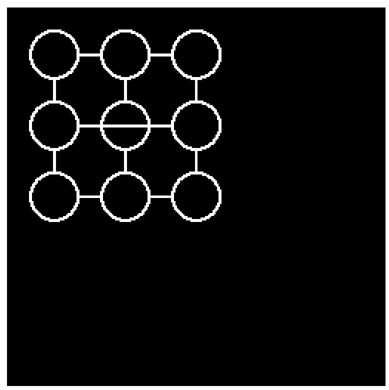
Solver timeout



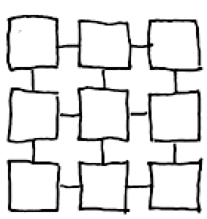


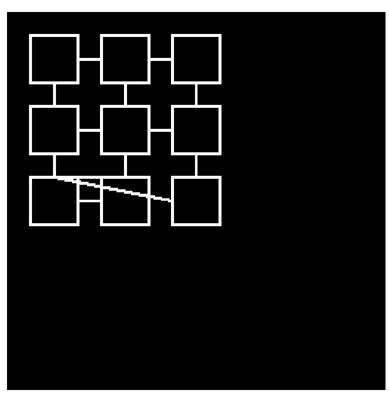
```
for (i < 3){
if (i > 0){
line(4*i + -3,-5*i + 12,2*i + 1,
arrow = False,solid = True);
rectangle(4*i + -4,-5*i + 10,6,-
}
circle(1,8)
}
```



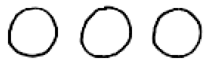


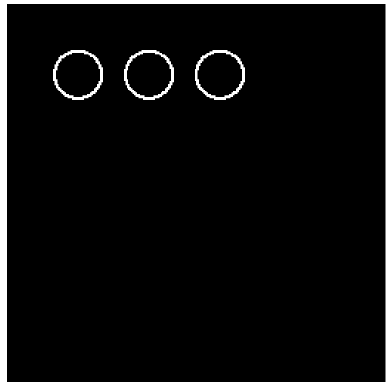
```
for (i < 3){
for (j < 3){
if (j > 0){
line(-3*j + 8,-3*i + 7,-3*j + 9,
arrow = False,solid = True);
line(-3*i + 7,-3*j + 8,-3*i + 7,
arrow = False,solid = True)
}
circle(-3*j + 7,-3*i + 7)
}
}
```



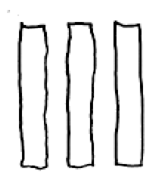


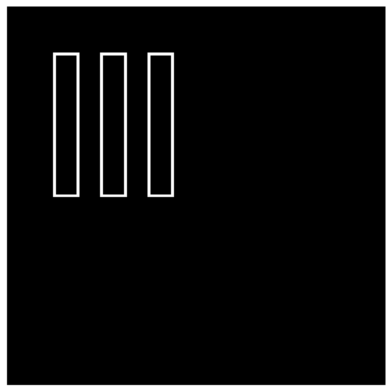
```
Solver timeout
```



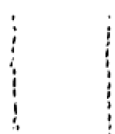


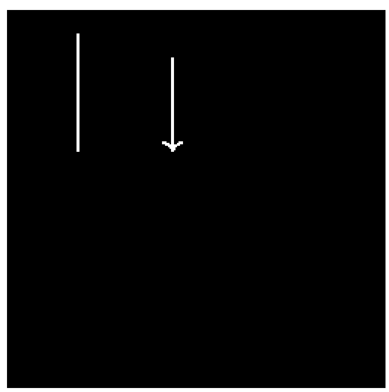
```
for (i < 3){
circle(-3*i + 7,1)
}
```



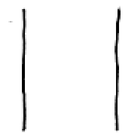


```
for (i < 3){
rectangle(-2*i + 4,0,-2*i + 5,6)
}
```

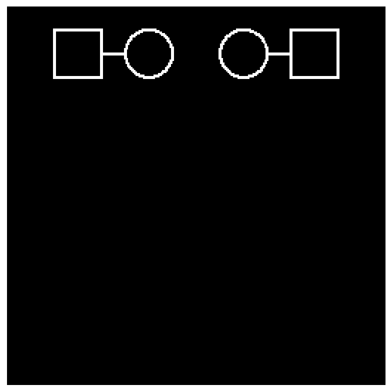




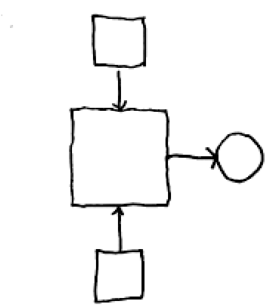
```
line(4,0,4,1,
arrow = False,solid = False);
line(0,0,0,5,
arrow = False,solid = False);
line(4,1,4,5,
arrow = False,solid = False)
```

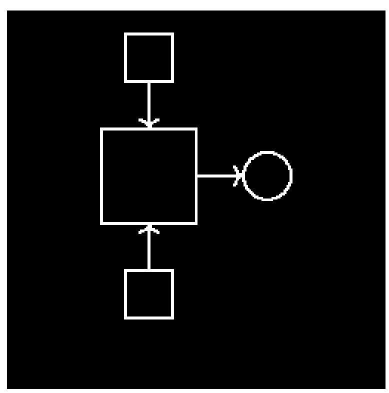


```
line(4,0,4,5,
arrow = False,solid = True);
line(0,0,0,5,
arrow = False,solid = True)
```

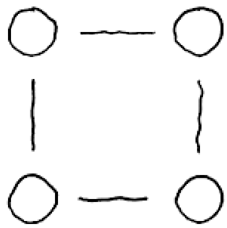


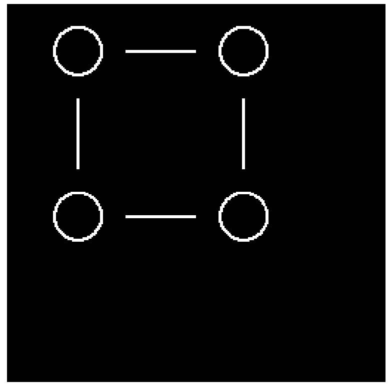
```
reflect(reflect(x = 12)){
circle(4,1);
line(9,1,10,1,
arrow = False,solid = True);
rectangle(0,0,2,2)
}
```



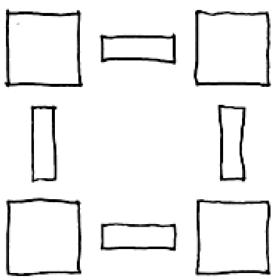


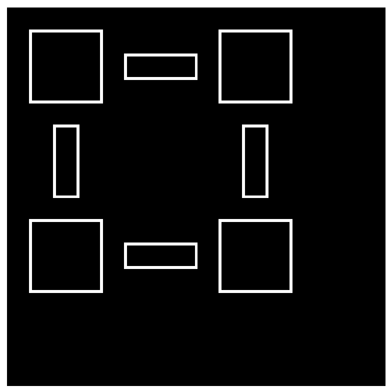
```
rectangle(0,4,4,8);
reflect(reflect(y = 12)){
circle(7,6);
line(2,2,2,4,
arrow = True,solid = True);
line(4,6,6,6,
arrow = True,solid = True);
rectangle(1,10,3,12)
}
```



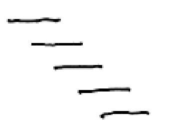


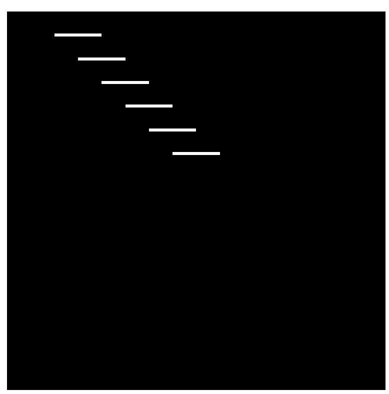
```
reflect(reflect(y = 9)){
line(3,8,6,8,
arrow = False,solid = True);
reflect(reflect(x = 9)){
circle(1,8);
line(1,3,1,6,
arrow = False,solid = True)
}
}
```



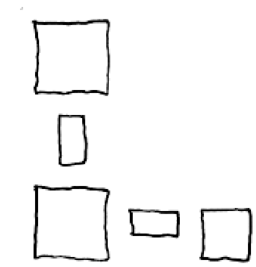


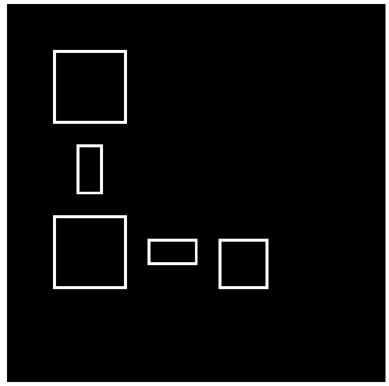
```
reflect(reflect(y = 11)){
rectangle(4,9,7,10);
reflect(reflect(x = 11)){
rectangle(1,4,2,7);
rectangle(8,8,11,11)
}
}
```



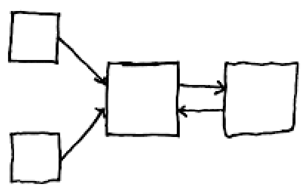


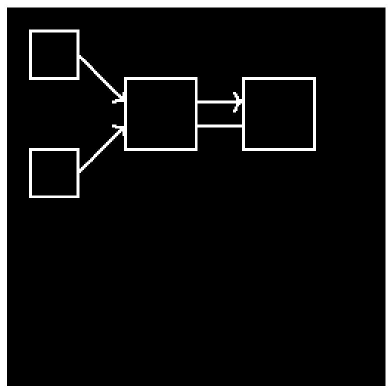
```
for (i < 4){
line(i,-1*i + 5,i + 2,-1*i + 5,
arrow = False,solid = True);
line(i + 2,-1*i + 3,i + 4,-1*i +
arrow = False,solid = True)
}
```



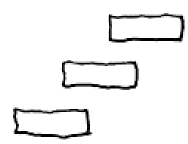


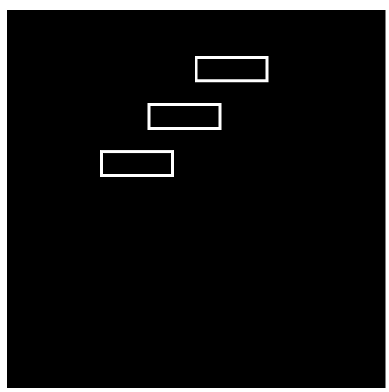
```
for (i < 3){
if (i > 0){
rectangle(3*i + 1,-1*i + 2,3*i +
rectangle(0,7*i + -7,3,7*i + -4)
}
rectangle(1,4,2,6)
}
```



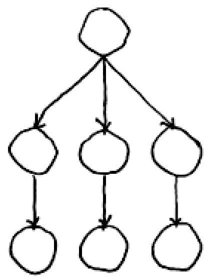


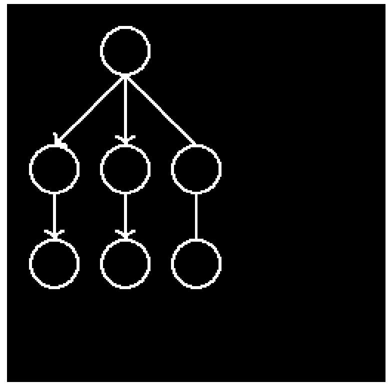
Solver timeout



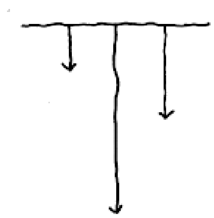


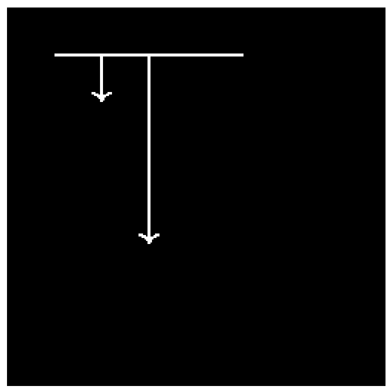
```
for (i < 3){
rectangle(-2*i + 4,-2*i + 4,-2*i
}
```



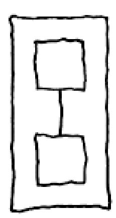


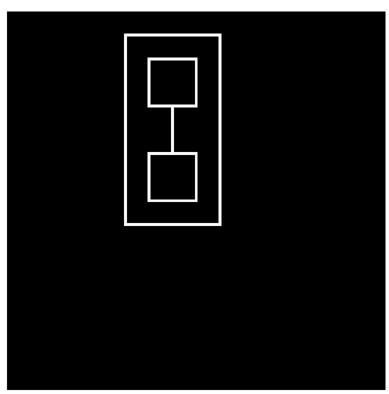
```
circle(4,10);
for (i < 3){
circle(-3*i + 7,5);
circle(-3*i + 7,1);
line(-3*i + 7,4,-3*i + 7,2,
arrow = True,solid = True);
line(4,9,-3*i + 7,6,
arrow = True,solid = True)
}
```



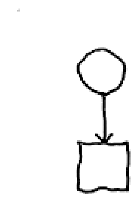


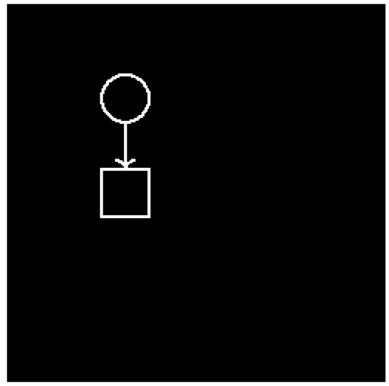
```
line(2,8,2,6,
arrow = True,solid = True);
line(6,8,6,4,
arrow = True,solid = True);
line(4,8,4,0,
arrow = True,solid = True);
line(0,8,8,8,
arrow = False,solid = True)
```



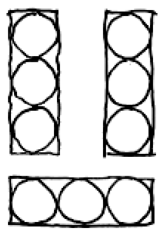


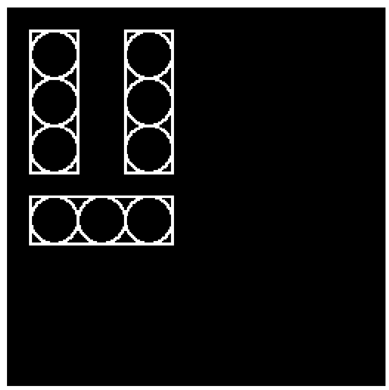
```
line(2,3,2,5,
arrow = False,solid = True);
rectangle(1,1,3,3);
rectangle(1,5,3,7);
rectangle(0,0,4,8)
```



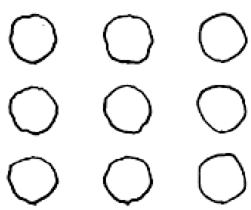


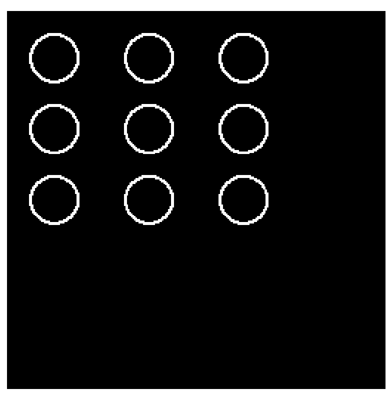
```
circle(1,5);
line(1,4,1,2,
arrow = True,solid = True);
rectangle(0,0,2,2)
```



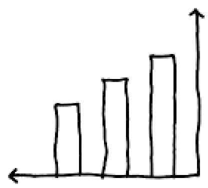


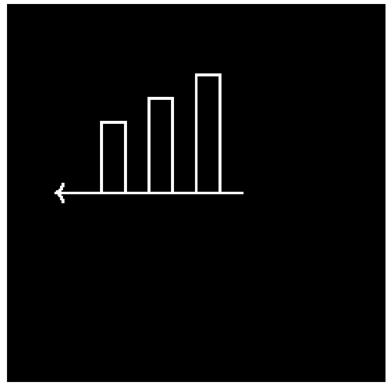
```
rectangle(0,0,6,2);
reflect(reflect(x = 6)){
rectangle(0,3,2,9);
for (i < 3){
circle(5,2*i + 4);
circle(2*i + 1,1)
}
}
```



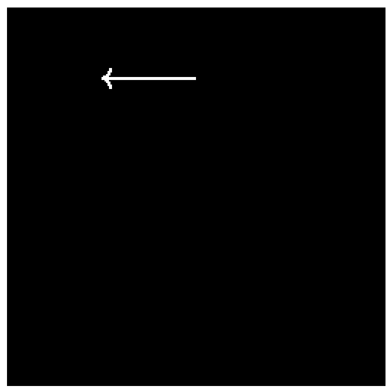


```
for (i < 3){
for (j < 3){
circle(-4*j + 9,-3*i + 7)
}
}
```

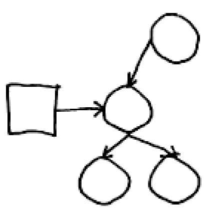


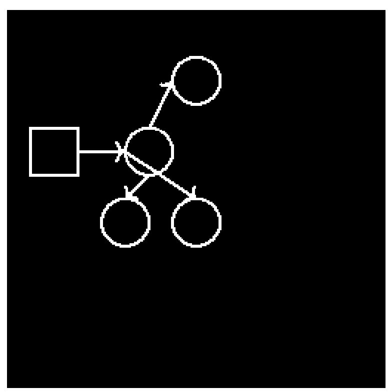


```
for (i < 3){
if (i > 0){
line(8,0,8*i + -8,7*i + -7,
arrow = True,solid = True)
}
rectangle(2*i + 2,0,2*i + 3,i +
}
```

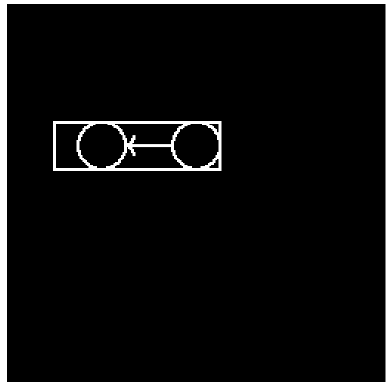



```
line(4,0,0,0,
arrow = False,solid = False)
```

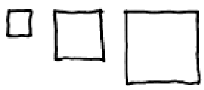


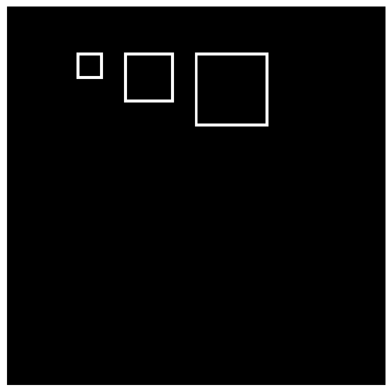


```
Solver timeout
```

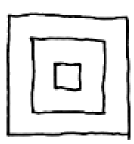


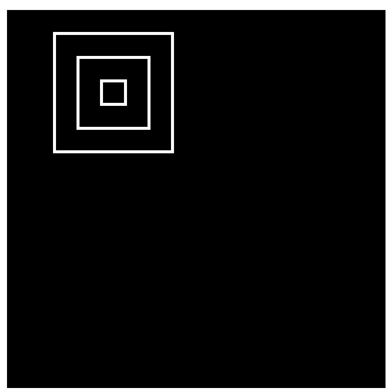
```
circle(2,1);
circle(6,1);
line(5,1,3,1,
arrow = True,solid = True);
rectangle(0,0,7,2)
```



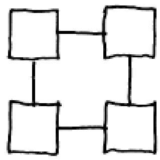


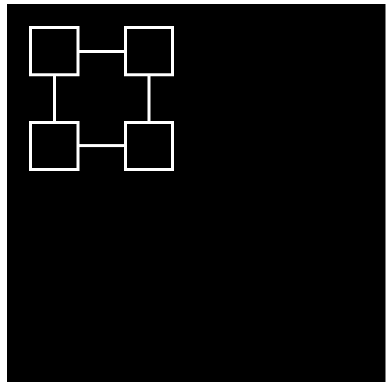
```
rectangle(5,0,8,3);
rectangle(2,1,4,3);
rectangle(0,2,1,3)
```



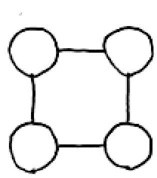


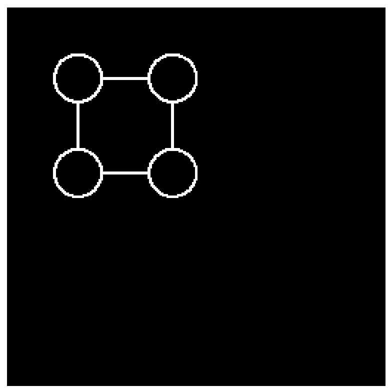
```
for (i < 3){
rectangle(-1*i + 2,-1*i + 2,i +
}
```



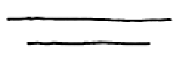


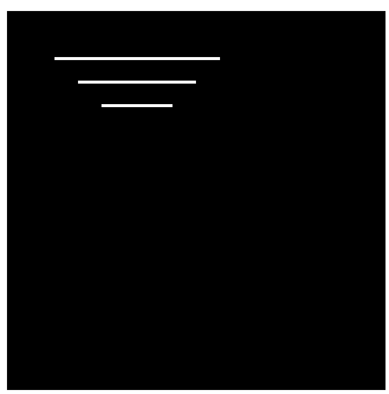
```
reflect(reflect(x = 6)){
line(5,2,5,4,
arrow = False,solid = True);
reflect(reflect(y = 6)){
line(2,1,4,1,
arrow = False,solid = True);
rectangle(4,4,6,6)
}
}
```



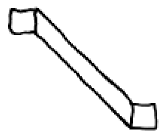


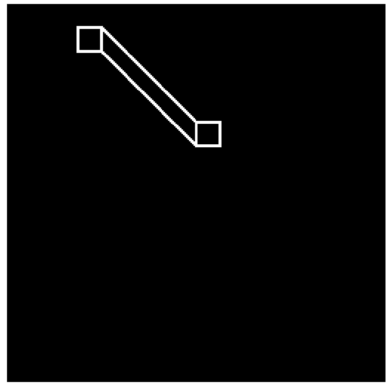
```
reflect(reflect(y = 6)){
line(2,5,4,5,
arrow = False,solid = True);
reflect(reflect(x = 6)){
circle(5,5);
line(1,2,1,4,
arrow = False,solid = True)
}
}
```



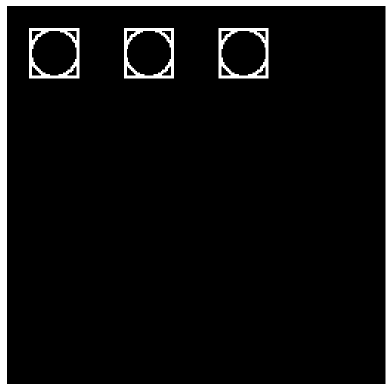


```
for (i < 3){
line(i,-1*i + 2,-1*i + 7,-1*i +
arrow = False,solid = True)
}
```

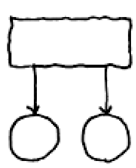


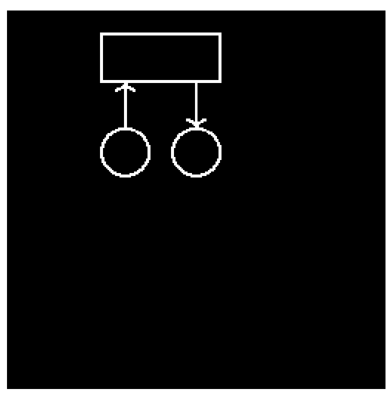


```
line(1,4,5,0,
arrow = False,solid = True);
line(1,5,5,1,
arrow = False,solid = True);
rectangle(5,0,6,1);
rectangle(0,4,1,5)
```

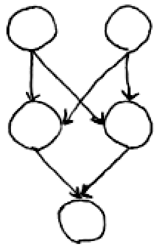


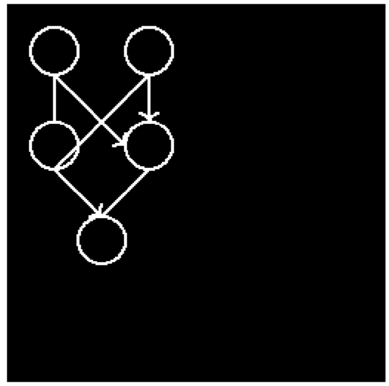
```
for (i < 3){
circle(4*i + 1,1);
rectangle(4*i,0,4*i + 2,2)
}
```



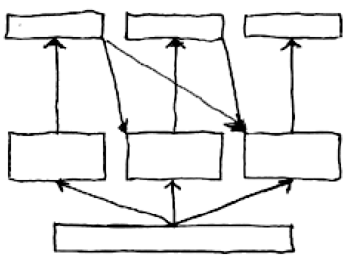


```
reflect(reflect(x = 5)){
circle(1,1);
line(4,4,4,2,
arrow = True,solid = True);
rectangle(0,4,5,6)
}
```



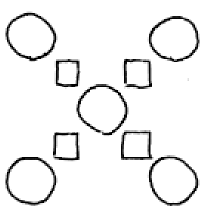


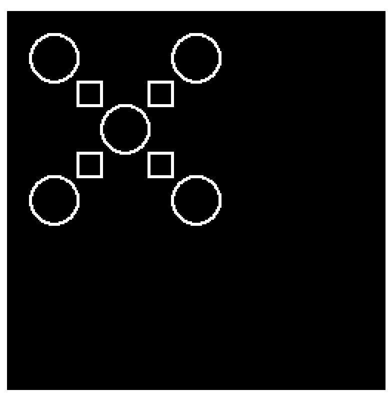
```
circle(3,1);
reflect(reflect(x = 6)){
for (i < 3){
if (i > 0){
circle(1,-4*i + 13);
line(5,-4*i + 12,-2*i + 7,-4*i +
arrow = True,solid = True)
}
line(1,8,4,5,
arrow = True,solid = True)
}
}
```



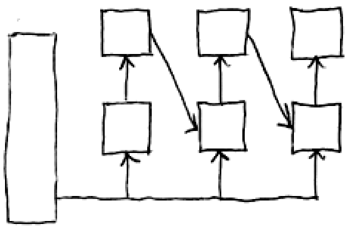
Sampled no finished traces.

```
Solver timeout
```



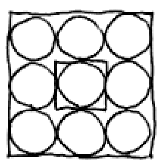


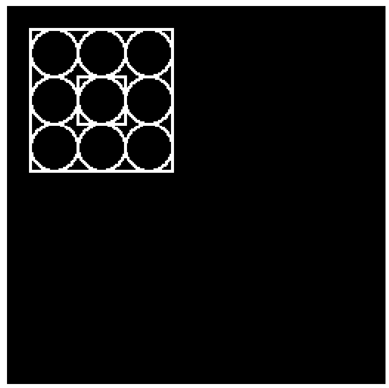
```
reflect(reflect(y = 8)){
for (i < 3){
if (i > 0){
rectangle(3*i + -1,2,3*i,3)
}
circle(3*i + 1,3*i + 1)
}
}
```



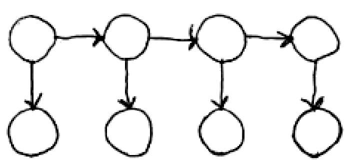
Sampled no finished traces.

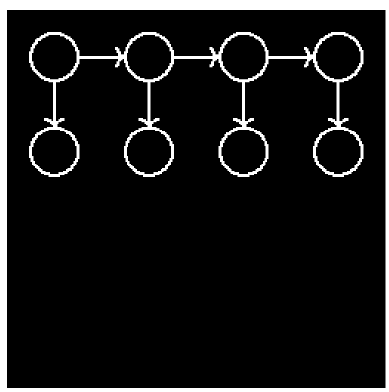
`Solver timeout`



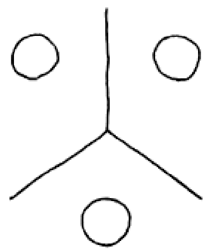


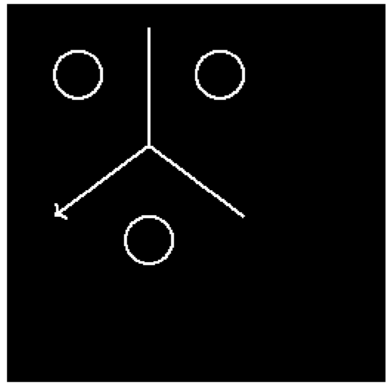
```
for (i < 3){
if (i > 0){
rectangle(-2*i + 4,-2*i + 4,2*i
}
for (j < 3){
circle(-2*i + 5,2*j + 1)
}
}
```



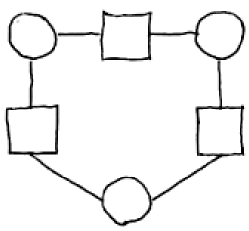


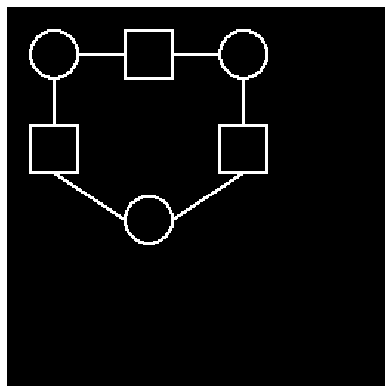
```
for (i < 4){
line(-4*i + 13,4,-4*i + 13,2,
arrow = True,solid = True);
for (j < 3){
if (j > 0){
circle(-4*i + 13,4*j + -3)
}
line(-4*j + 10,5,-4*j + 12,5,
arrow = True,solid = True)
}
}
```



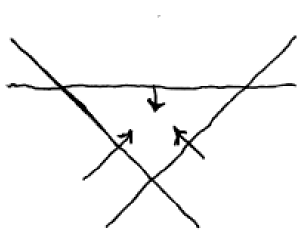


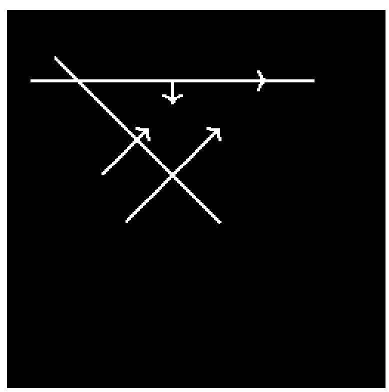
```
circle(4,1);
reflect(reflect(x = 8)){
circle(1,8);
line(4,5,8,2,
arrow = False,solid = True);
line(4,5,4,10,
arrow = False,solid = True)
}
```



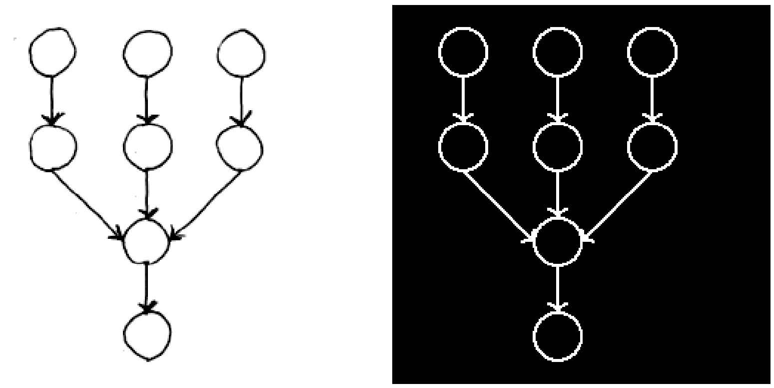


Solver timeout

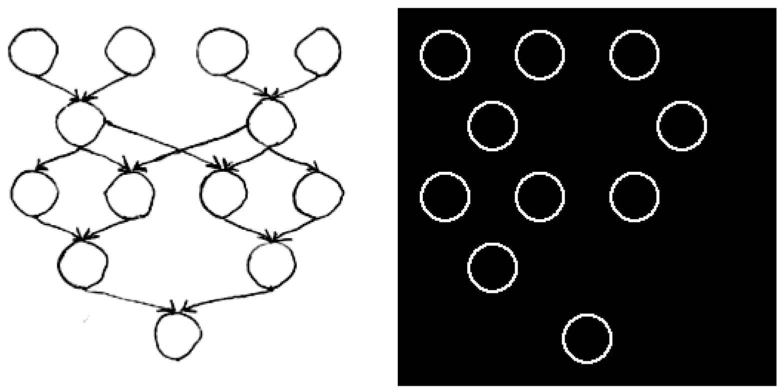




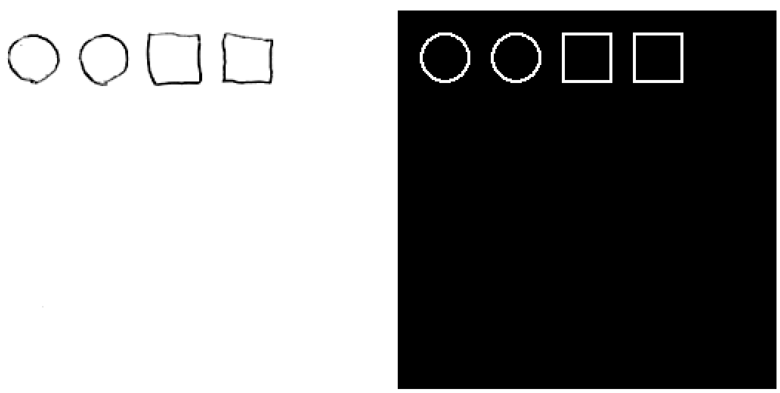
```
line(0,6,12,6,
arrow = False,solid = True);
line(6,6,6,5,
arrow = True,solid = True);
line(8,3,7,4,
arrow = True,solid = True);
line(3,2,5,4,
arrow = True,solid = True);
reflect(reflect(x = 12)){
line(4,0,12,8,
arrow = False,solid = True)
}
```



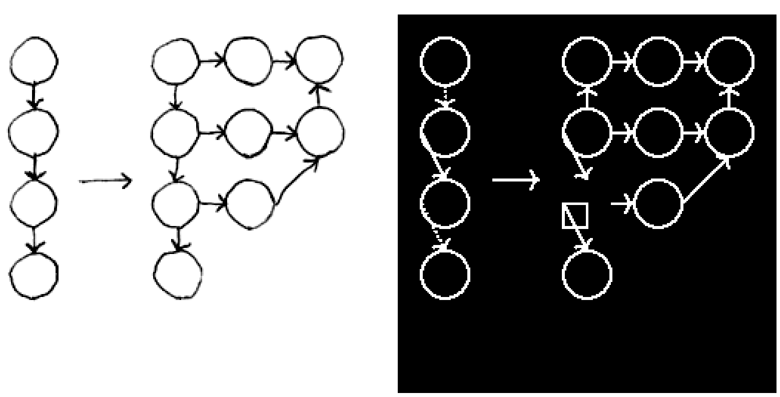
Solver timeout



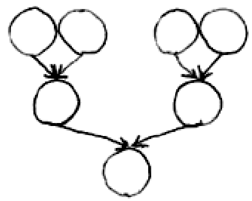
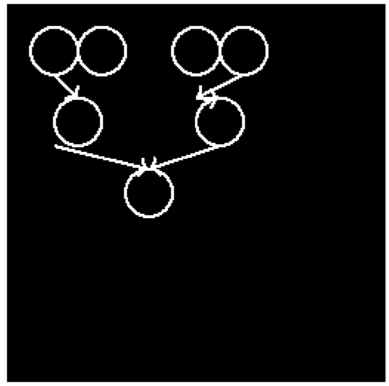
Solver timeout



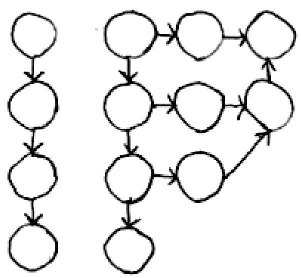
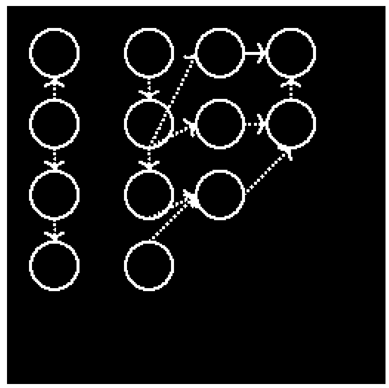
```
circle(1,1);
circle(4,1);
rectangle(9,0,11,2);
rectangle(6,0,8,2)
```



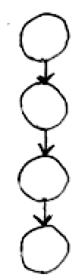
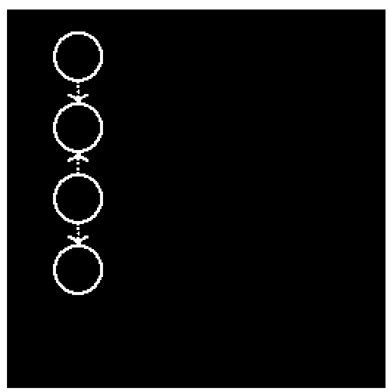
Solver timeout

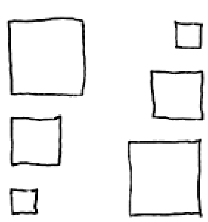
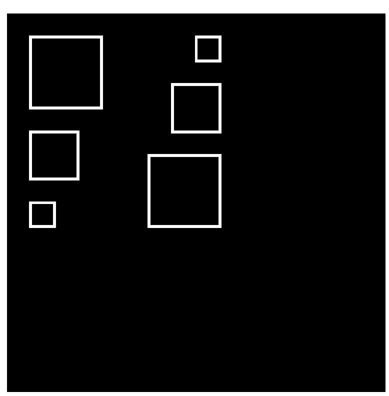
Solver timeout

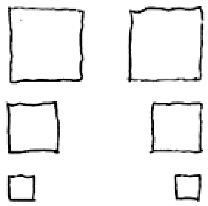
Solver timeout

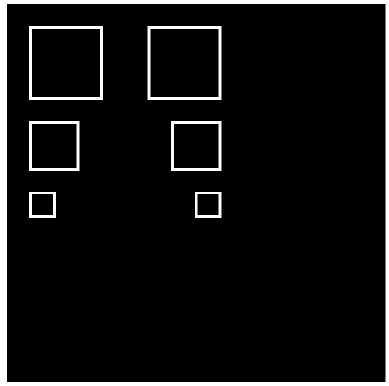



```
for (i < 4){
if (i > 0){
line(1,-3*i + 12,1,-3*i + 11,
arrow = True,solid = True)
}
circle(1,-3*i + 10)
}
```

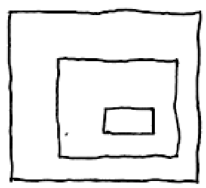



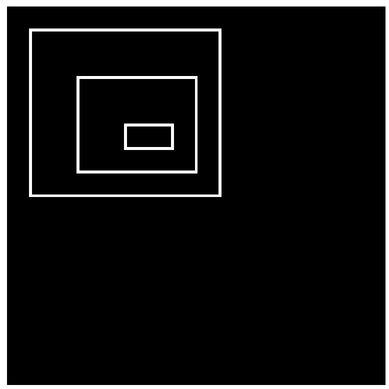
Solver timeout



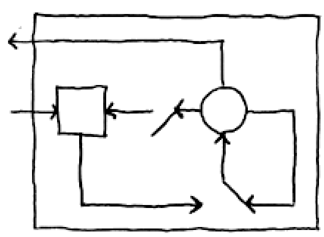


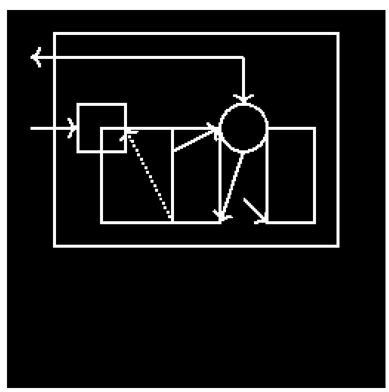
```
reflect(reflect(x = 8)){
rectangle(0,5,3,8);
rectangle(0,2,2,4);
rectangle(0,0,1,1)
}
```



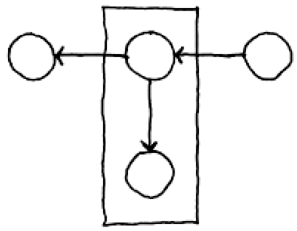


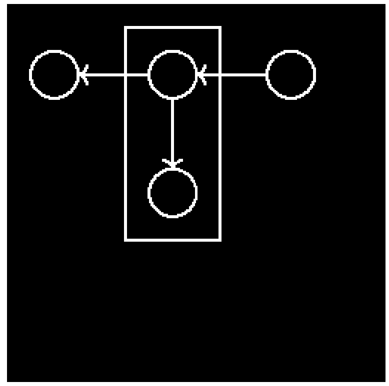
```
for (i < 3){
rectangle(-2*i + 4,-1*i + 2,i +
}
```



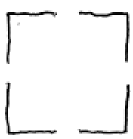


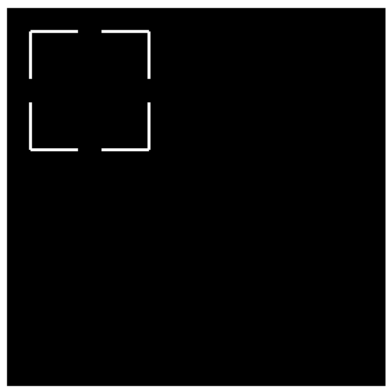
```
Solver timeout
```



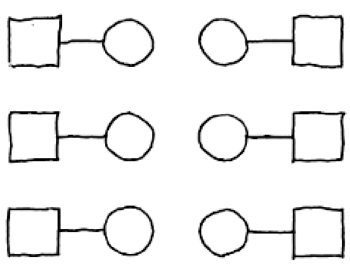


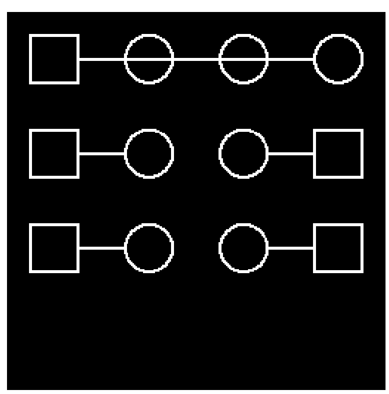
```
line(6,6,6,3,
arrow = True,solid = True);
for (i < 3){
if (i > 0){
circle(-5*i + 16,7);
circle(-5*i + 11,5*i + -3);
line(-5*i + 15,7,-5*i + 12,7,
arrow = True,solid = True)
}
rectangle(4,0,8,9)
}
```



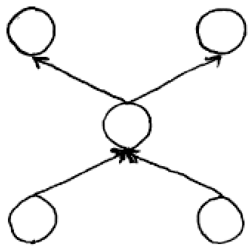


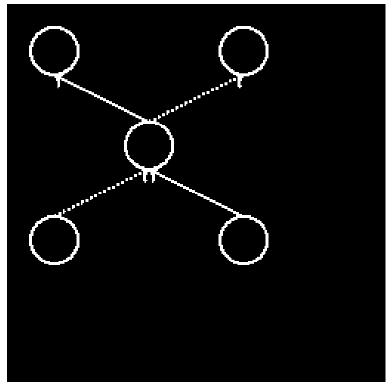
```
reflect(reflect(x = 5)){
reflect(reflect(y = 5)){
line(5,3,5,5,
arrow = False,solid = True);
line(3,5,5,5,
arrow = False,solid = True)
}
}
```



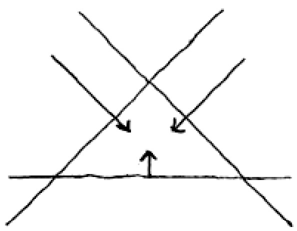


```
reflect(reflect(x = 14)){
for (i < 3){
circle(9,-4*i + 9);
line(10,-4*i + 9,12,-4*i + 9,
arrow = False,solid = True);
rectangle(0,-4*i + 8,2,-4*i + 10
}
}
```



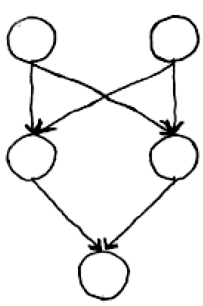


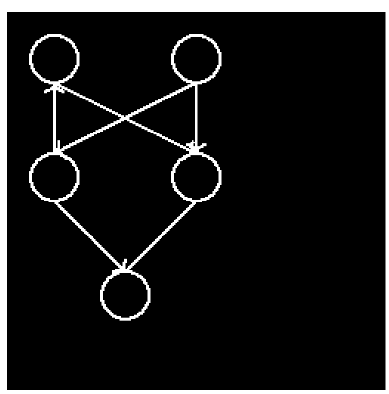
```
reflect(reflect(x = 10)){
for (i < 3){
if (i > 0){
line(4*i + -3,4*i + -2,4*i + 1,4
arrow = True,solid = True)
}
circle(4*i + 1,4*i + 1)
}
}
```



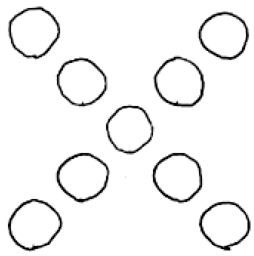
Sampled no finished traces.

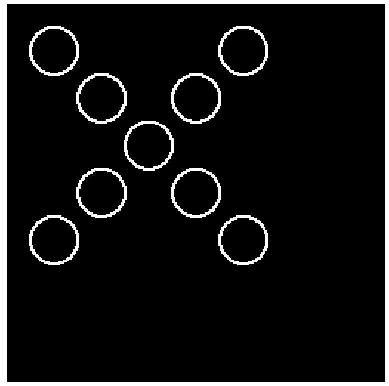
```
line(0,2,12,2,
arrow = False,solid = True);
line(6,2,6,3,
arrow = True,solid = True);
reflect(reflect(x = 12)){
line(0,0,9,9,
arrow = False,solid = True);
line(10,7,7,4,
arrow = True,solid = True)
}
```



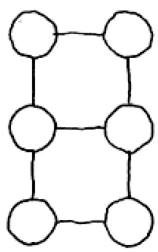


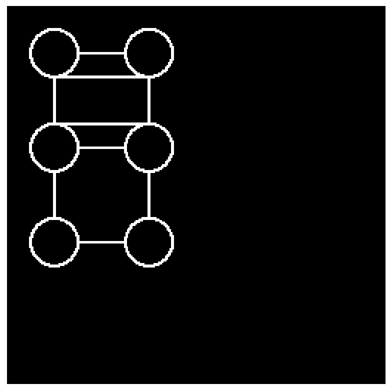
```
reflect(reflect(x = 8)){
circle(4,1);
for (i < 3){
if (i > 0){
circle(7,-5*i + 16);
line(-6*i + 13,10,7,7,
arrow = True,solid = True)
}
line(1,5,4,2,
arrow = True,solid = True)
}
}
```



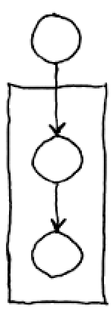


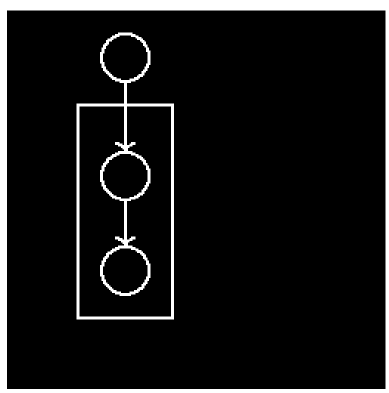
```
reflect(reflect(x = 10)){
circle(1,1);
for (i < 4){
circle(-2*i + 7,2*i + 3)
}
}
```



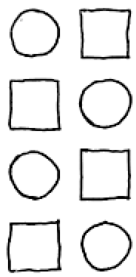


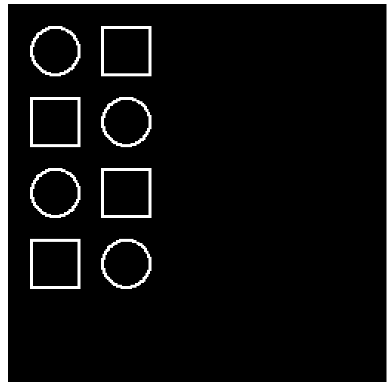
```
reflect(reflect(x = 6)){
for (i < 3){
if (i > 0){
line(1,-4*i + 10,1,-4*i + 12,
arrow = False,solid = True)
}
circle(5,-4*i + 9);
line(2,-4*i + 9,4,-4*i + 9,
arrow = False,solid = True)
}
}
```





```
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)
}
```





```
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
}
```

References

[1] Samuel J Gershman and David M Blei. A tutorial on bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12, 2012.