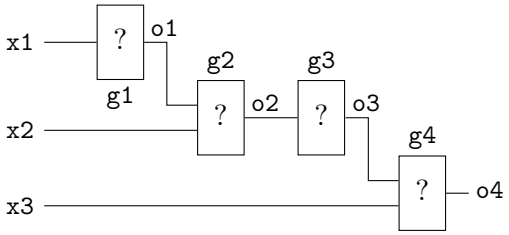
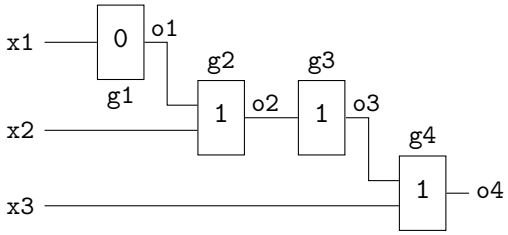


(a) A Boolean circuit with four gates



(b) The unique solution

Figure 1: A problem instance and its unique solution

1 Gates to truth

In this problem, we are given the layout of a Boolean circuit with unary and binary gates, yet we have to decide which logical operations the gates should perform. For each unary gate, we have to choose between the identity function and negation, and between logical AND and OR for the binary gates. A partial specification requires specific output values for some truth assignments of the input variables, while the output can be arbitrary for the remaining truth assignments.

Example

The following facts describe the problem instance shown in Figure 1(a) by facts, which can be given to an ASP or Prolog system to be used for solving:

```
variable(x1).
variable(x2).
variable(x3).
unary(g1).
binary(g2).
unary(g3).
binary(g4).
input(g1,1,x1).
output(g1,o1).
input(g2,1,o1).
input(g2,2,x2).
output(g2,o2).
input(g3,1,o2).
output(g3,o3).
input(g4,1,o3).
input(g4,2,x3).
output(g4,o4).
result(o4).
entry(1,x1,0).
entry(1,x2,0).
entry(1,x3,0).
value(1,o4,1).
entry(2,x1,0).
entry(2,x2,1).
entry(2,x3,0).
```

```
value(2,o4,0).
entry(3,x1,1).
entry(3,x2,1).
entry(3,x3,0).
value(3,o4,0).
entry(4,x1,1).
entry(4,x2,0).
entry(4,x3,1).
value(4,o4,1).
```

These facts declare three input variables (`x1`, `x2`, and `x3`) and four gates (`g1`, `g2`, `g3`, and `g4` whose output values are denoted by `o1`, `o2`, `o3`, and `o4`) of the Boolean circuit. The unary gates `g1` and `g3` have a single input each, while the binary gates `g2` and `g4` take two inputs. Function values computed by the Boolean circuit are read off from the output `o4` of gate `g4`.

Four truth assignments require specific output values to be computed by the Boolean circuit:

x1	x2	x3	o4
0	0	0	1
0	1	0	0
1	1	0	0
1	0	1	1

For the remaining (four) truth assignments, which are not listed in the table, the output value `o4` can be arbitrary.

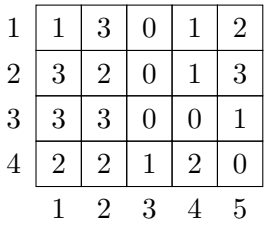
The (unique) solution for this example, as shown in Figure 1(b), determines the logical operations performed by the four gates. It can be represented as follows by ASP or Prolog atoms:

ASP: `function(g1,0). function(g2,1). function(g3,1). function(g4,1).`

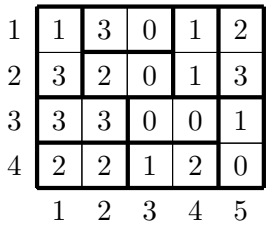
Prolog: `function([(g1,0), (g2,1), (g3,1), (g4,1)]).`

That is, the unary gate `g1` computes the identify function, and `g3` provides the negation of its input. Both binary gates `g2` and `g4` compute the logical OR to match the partial specification.

In general, there could be several Boolean circuits that are compatible with a given partial specification. Each of our test instances has a unique solution though, which may help as a quick criterion for checking the correctness of your problem encoding.



(a) A 4×5 grid



(b) The unique solution

Figure 2: A problem instance and its unique solution

2 Tile them all

Consider a rectangular 4×5 grid such that each of its cells contains a number: 0, 1, 2, or 3. The goal is to group horizontally or vertically adjacent cells into ten disjoint pairs such that all ten of the combinations $\{0,1,2,3\} \times \{0,1,2,3\}$ are represented by a pair, also called tile.

Example

You can choose between two alternative representations of a rectangular 4×5 grid, such as the one shown in Figure 2(a), taking the input format you consider more convenient:

ASP:

```
cell(1,1,1).
cell(1,2,3).
cell(1,3,0).
cell(1,4,1).
cell(1,5,2).
cell(2,1,3).
cell(2,2,2).
cell(2,3,0).
cell(2,4,1).
cell(2,5,3).
cell(3,1,3).
cell(3,2,3).
cell(3,3,0).
cell(3,4,0).
cell(3,5,1).
cell(4,1,2).
cell(4,2,2).
cell(4,3,1).
cell(4,4,2).
cell(4,5,0).
```

Prolog:

```
cell([
[1, 3, 0, 1, 2],
```

```
[3, 2, 0, 1, 3],
[3, 3, 0, 0, 1],
[2, 2, 1, 2, 0]
]).
```

The (unique) solution for this example, as shown in Figure 2(b), can likewise be represented in one of the following alternative formats:

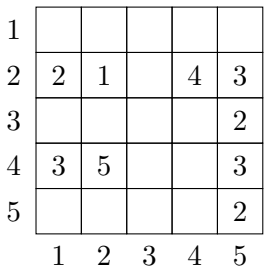
ASP:

```
tile(2,4,1,4).
tile(4,2,4,1).
tile(1,2,1,3).
tile(2,2,2,3).
tile(3,1,3,2).
tile(3,3,3,4).
tile(4,1,4,2).
tile(4,3,4,4).
tile(1,1,2,1).
tile(1,4,2,4).
tile(1,5,2,5).
tile(3,5,4,5).
tile(4,5,3,5).
tile(2,5,1,5).
tile(2,1,1,1).
tile(4,4,4,3).
tile(3,4,3,3).
tile(3,2,3,1).
tile(2,3,2,2).
tile(1,3,1,2).
```

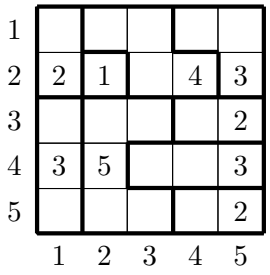
Prolog:

```
tile([
[(1,0),  (0,1),  (0,-1), (1,0),  (1,0) ],
[(-1,0), (0,1),  (0,-1), (-1,0), (-1,0)],
[(0,1),  (0,-1), (0,1),  (0,-1), (1,0) ],
[(0,1),  (0,-1), (0,1),  (0,-1), (-1,0)]
]).
```

Both solution representations are symmetric in the sense that each cell refers to its adjacent neighbor in a tile, and it is in turn also referred to by this neighbor. Again, each of our instances has a unique solution, which may help for quickly screening the correctness of your problem encoding.



(a) A 5×5 grid



(b) The unique solution

Figure 3: A problem instance and its unique solution

3 Area fifty won

Consider a 5×5 grid such that some of its cells contain positive numbers and the remaining cells are empty. The goal is to partition the grid into as many disjoint areas as there cells with positive numbers, where each area consists of exactly one cell with a positive number N and $N-1$ empty cells. Moreover, the cells of each area must form a connected region, i.e., there is a path via horizontally or vertically adjacent neighbors between any two cells belonging to the same area.

Example

You can choose between two alternative representations of a 5×5 grid, such as the one shown in Figure 3(a), taking the input format you consider more convenient:

ASP:

```
cell(1,1,0).
cell(1,2,0).
cell(1,3,0).
cell(1,4,0).
cell(1,5,0).
cell(2,1,2).
cell(2,2,1).
cell(2,3,0).
cell(2,4,4).
cell(2,5,3).
cell(3,1,0).
cell(3,2,0).
cell(3,3,0).
cell(3,4,0).
cell(3,5,2).
cell(4,1,3).
cell(4,2,5).
cell(4,3,0).
cell(4,4,0).
cell(4,5,3).
cell(5,1,0).
cell(5,2,0).
```

```
cell(5,3,0).
cell(5,4,0).
cell(5,5,2).
```

Prolog:

```
cell([
[0, 0, 0, 0, 0],
[2, 1, 0, 4, 3],
[0, 0, 0, 0, 2],
[3, 5, 0, 0, 3],
[0, 0, 0, 0, 2]
]).
```

Note that empty cells are indicated by associating them with the number `0` in both representations.

The (unique) solution for this example, as shown in Figure 3(b), can likewise be represented in one of the following alternative formats:

ASP:

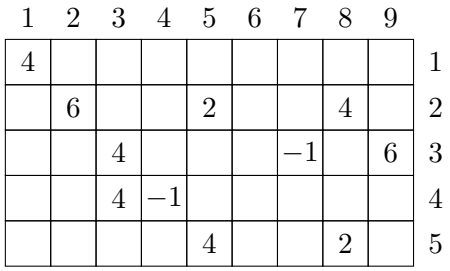
```
area(2,1,2,1).
area(2,2,2,2).
area(2,4,2,4).
area(2,5,2,5).
area(3,5,3,5).
area(4,1,4,1).
area(4,2,4,2).
area(4,5,4,5).
area(5,5,5,5).
area(5,5,5,4).
area(4,5,4,4).
area(4,2,3,2).
area(4,2,5,2).
area(4,1,3,1).
area(4,1,5,1).
area(3,5,3,4).
area(2,5,1,5).
area(2,4,2,3).
area(2,1,1,1).
area(2,4,1,3).
area(2,5,1,4).
area(4,2,5,3).
area(4,2,3,3).
area(4,5,4,3).
area(2,4,1,2).
```

Prolog:

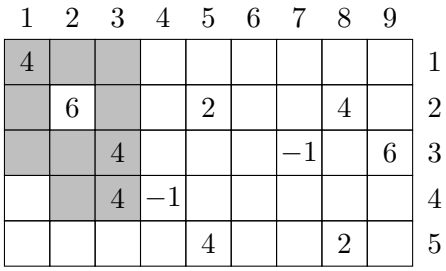
```
area([
[(1,1), (2,1)],
[(1,2), (1,3), (2,3), (2,4)],
[(1,4), (1,5), (2,5)],
```

```
[(2,2)],
[(3,1), (4,1), (5,1)],
[(3,2), (3,3), (4,2), (5,2), (5,3)],
[(3,4), (3,5)],
[(4,3), (4,4), (4,5)],
[(5,4), (5,5)]
]).
```

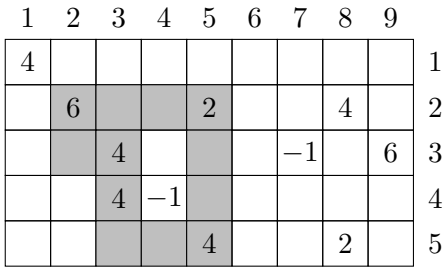
Atoms of the form `area`(x_1,y_1,x_2,y_2) in the first representation express that the cell (x_2,y_2) belongs to the area with the numbered cell (x_1,y_1). That is, the cells (x_1,y_1) and (x_2,y_2) share the same sublist in the second representation. Again, each of our instances has a unique solution, which may help for quickly screening the correctness of your problem encoding.



(a) A 9×5 grid



(b) A non-optimal solution



(c) An optimal solution

Figure 4: A problem instance and two solutions of different quality

4 Fence for fans

Consider a $W \times H$ grid of width W and height H such that, for each cell c, a non-negative cost $\text{cost}(c)$ and a (possibly negative) reward $\text{reward}(c)$ is given. The goal is to set up a fence, forming a non-empty and non-simple cycle, which must not include any cell with a negative reward ($\text{reward}(c) < 0$ is not permitted for cells c belonging to the cycle) and maximizes the following quality function:

$$\sum_{\text{cell } c \text{ belongs to the cycle}} (\text{reward}(c) - \text{cost}(c)) \tag{1}$$

Moreover, given limits on the length of the cycle, i.e., the number of contained cells, and the budget for setting up the cycle, i.e., the sum of costs $\text{cost}(c)$ over the cells c belonging to the cycle, must not be exceeded.

Example

For the 9×5 grid shown in Figure 4(a), we assume a uniform cost of 1 for each cell and display non-zero rewards within the cells (i.e., the empty cells have reward 0). The two alternative input formats representing this problem instance are as follows:

ASP:

```
width(9).
height(5).
cost(1,1,1).
cost(2,1,1).
```

```
cost(3,1,1).
% [...]
cost(7,5,1).
cost(8,5,1).
cost(9,5,1).
reward(1,1,4).
reward(2,1,0).
reward(3,1,0).
% [...]
reward(2,2,6).
% [...]
reward(5,2,2).
% [...]
reward(8,2,4).
% [...]
reward(3,3,4).
% [...]
reward(7,3,-1).
% [...]
reward(9,3,6).
% [...]
reward(3,4,4).
reward(4,4,-1).
% [...]
reward(5,5,4).
% [...]
reward(8,5,2).
reward(9,5,0).
length(12).
budget(12).
```

Prolog:

```
width(9).
height(5).
cost([
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1, 1]
]).
reward([
[ 4, 0, 0, 0, 0, 0, 0, 0, 0],
[ 0, 6, 0, 0, 2, 0, 0, 4, 0],
[ 0, 0, 4, 0, 0, 0,-1, 0, 6],
[ 0, 0, 4,-1, 0, 0, 0, 0, 0],
[ 0, 0, 0, 0, 4, 0, 0, 2, 0]
]).
length(12).
budget(12).
```

The non-optimal solution in Figure 4(b) includes 10 cells of cost 1 each, so that the length and budget limits are respected. It collects rewards adding up to 12, so that the solution quality calculated according to (1) is $12 - 10 = 2$. However, beware of non-optimal solutions, as we are not interested into them! Rather, we want to find some optimal solution like the one in Figure 4(c). It includes 12 cells and collects rewards adding up to 20, so that the quality of this solution is $20 - 12 = 8$. The following two alternative formats can be used to represent this optimal solution:

ASP:

```
fence(2,2,2,3).
fence(5,2,5,3).
fence(3,3,3,4).
fence(5,3,5,4).
fence(3,4,3,5).
fence(5,4,5,5).
fence(2,2,3,2).
fence(3,2,4,2).
fence(4,2,5,2).
fence(2,3,3,3).
fence(3,5,4,5).
fence(4,5,5,5).
fence(5,5,4,5).
fence(4,5,3,5).
fence(3,3,2,3).
fence(5,2,4,2).
fence(4,2,3,2).
fence(3,2,2,2).
fence(5,5,5,4).
fence(3,5,3,4).
fence(5,4,5,3).
fence(3,4,3,3).
fence(5,3,5,2).
fence(2,3,2,2).
```

Prolog:

```
fence([(2,2), (2,3), (3,3), (3,4), (3,5), (4,5),
       (5,5), (5,4), (5,3), (5,2), (4,2), (3,2)]).
```

Atoms of the form `fence(`x_1`,`y_1`,`x_2`,`y_2`)` in the first representation express that the fence connects the horizontally or vertically adjacent cells `(`x_1`,`y_1`)` and `(`x_2`,`y_2`)`, where the format is symmetric and includes the atom `fence(`x_2`,`y_2`,`x_1`,`y_1`)` as well. The list in the second representation provides the cells of the cycle in order, where the first cell is arbitrary but the last cell must be horizontally or vertically adjacent to it in order to close the cycle. Note that optimal solutions to our instances are usually not unique, and your problem encoding must yield some of the optimal solutions (i.e., finding one optimal solution per instance is sufficient).

Harder version

Once you are done with the "simple" problem above, you can go for a harder version that extends the quality function (1) by additionally incorporating the rewards for cells enclosed by the fence.

Such rewards can be negative, as it happens for the formerly optimal solution in Figure 4(c) whose quality reduces from 8 to 7. On the other hand, the non-optimal solution in Figure 4(b) is such that the fence encloses a cell with the reward 6, which boosts the quality from 2 to 8. Indeed, the solution shown in Figure 4(b) is one of several optimal solutions for the harder problem version.