Matthew Roberson
CSE 494 Final Report

Problem Description

Light Up, a binary-determination logic puzzle, is played on a rectangular grid containing white and black cells. The puzzle is solved once lights bulbs are placed in white cells such that no two bulbs shine on each other, and the entire grid is lit up. The following rules apply:

- each bulb sends rays of light both vertically and horizontally
- each bulb illuminates its entire row/column unless its light is blocked by a black cell
- a black cell may contain a number 0-4 which indicates the number of bulbs that must be placed adjacent to its four sides
- an unnumbered black cell may have any number of bulbs around it
- bulbs placed diagonally adjacent to any black cell do not contribute to its bulb count

Kuromasu, a binary-determination logic puzzle, is played on a rectangular grid. Cells are either white or black, with some white cells containing numbers. The goal is to determine which cells are white and which cells are black. The following rules apply:

- each number (contained in a white cell) represents the number of white cells that can be seen from that cell, including itself. A cell is seen if it is contained in the same row or column and there is no black cell between them in that row or column.
- numbered cells must be white
- no two black cells may be horizontally or vertically adjacent
- all white cells must be connected horizontally or vertically

The goal of this project is to design a solver for each type of puzzle.

Implementation

Kuromasu

As input, Kuromasu takes a list of triples representing numbered white cells. Each triple contains the XY coordinate of a white cell, as well as the numbered value contained within that cell. This list can be supplied either within the code, or imported via the

command line. For convenience, the input has been included in the LPARSE file. For example,

num(X,Y,N)

where X,Y represent the corresponding XY coordinate and N the number
As output, Kuromasu produces a list of pairs representing black cells. Each pair represents the location of a black cell within the grid:

black(X,Y)

(see section on Sample Puzzles for sample Kuromasu puzzle and solution)

In Kuromasu, cells are either white or black. White cells, and only white cells, may contain numbers. Therefore to begin, a subset of grid cells must be designated white, and correspondingly, the rest must be designated black:

```
%generate white cells
{cell(w,X,Y)} :- row(X), col(Y).

%define black cells
cell(b,X,Y) :- not cell(w,X,Y), row(X), col(Y).
```

Also, numbered cells must be white. As the "num" variable does not refer to color, a rule must be added that designates a numbered cell as a white cell. A constraint can be added such that if an answer set contains a numbered cell that is not white, that answer set is removed:

```
%numbered cells must be white
:- not cell(w,X,Y), num(X,Y,N).
```

Also, no two black cells may be horizontally or vertically adjacent. Therefore, it is necessary to define adjacency and introduce a new variable. Two cells (X,Y) and (R,S) are adjacent if and only if:

$$|X-R| + |Y-S| = 1$$

Adjacency is therefore defined as either horizontal or vertical, i.e. orthogonal adjacency. Diagonal adjacency is not precluded. In order to remove possible solutions that contain

horizontal or vertical pairs of black cells, it is necessary to add a constraint that removes those possible solutions. The following is the definition of adjacency with the corresponding constraint removing horizontally or vertically adjacent black cells:

```
%define adjacency
adj(X,Y,R,S) :- row(X;R), col(Y;S), abs(X-R)+abs(Y-S) == 1.

%no two black cells can be horizontally or vertically adjacent
:- cell(b,X,Y), cell(b,R,S), adj(X,Y,R,S), row(X;R), col(Y;S).
```

As each numbered cell represents a group of white cells, it is also necessary to define connectedness such that two white cells are connected if and only if they are contained in either the same row or column and do not contain a black cell between them. Each cell is also connected to itself, as it is stated that each numbered group must contain the numbered cell. The following two rules define connectedness:

```
%define connectedness for numbered cells (only white cells are numbered)
connect_cell(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
connect_cell(X,Y,R,S) :- connect_cell(X,Y,T,U), adj(R,S,T,U),or(X==R,Y==S),
                                    cell(w,R,S), row(X;R;T), col(Y;S;U).
```

Note, however, that within the definition of *connect_cell* there does not contain an input parameter representing color. As connectedness is only defined for white cells, it would be superfluous to include a variable representing color as in input parameter for *connect_cell*. But, color must be made explicit within the right-hand side of the definition in order to define connectedness as only applicable to white cells,.

Connectedness is also defined for white cells such that all white cells must be horizontally or vertically connected. This definition of connectedness is more general than the definition of connectedness for numbered cells and does not need to include the constraint within the definition such that white cells are connected if and only if they are contained in the same row or column. Therefore, the second definition of connectedness redefines the previous definition but does not include the previously mentioned constraint:

```
%define connectedness for white cells
white_connect(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
```

```
white_connect(X,Y,R,S) :- white_connect(X,Y,T,U), adj(R,S,T,U), cell(w,R,S),
                          row(X;R;T), col(Y;S;U).
```

In order to ensure that all white cells are connected either horizontally or vertically, it must be the case that every white cell be connected to every other white cell. Therefore, a constraint must be added such that answer sets that contain two white cells such that they are not connected, as defined by *white_connect*, are removed. The constraint is represented by the following rule:

```
%make sure every white cell is reachable from any given white cell
:- not white_connect(X,Y,R,S), cell(w,X,Y), cell(w,R,S), row(X;R), col(Y;S).
```

The number contained in each numbered cell represents the number of white cells that can be seen from that cell including itself. Therefore, with respect to each numbered cell, it must be contained within a group of that same number. Such a rule is possible by combining a choice formula with a cardinality expression:

```
%every numbered cell should be seen by N white cells
seen(X,Y) :- N{connect_cell(X,Y,R,S):row(R):col(S)}N, num(X,Y,N).
:- not seen(X,Y), num(X,Y,N).
```

The above constraint is added to ensure that all numbered cells are included in the answer set.

A final definition is also included in order to format the output, as specified earlier.

```
black(X,Y) :- cell(b,X,Y), row(X), col(Y).
```

Light Up

As input, Light Up takes a list composed of triples and pairs representing black cells. Some black cells are numbered, while some are not. Each triple represents a single XY coordinate, as well as the numbered value contained within that cell, while each pair represents a single XY coordinate with no numbered value. Numbered black cells are represented by the variable:

```
black(X,Y,N)
```

Unnumbered black cells are represented by the variable:

black(X,Y)

As output, Light Up produces a list of pairs representing the placement of lightbulbs within the grid:

lightbulb(X,Y)

(See section on Sample Puzzles for sample Light Up puzzle and solution)

As Light Up is a similar style of puzzle to Kuromasu, both puzzles utilize similar concepts, such as connectedness and adjacency. Therefore, these rules will be mentioned rather than explicitly defined, but should be considered to be implemented the same as above unless mentioned.

In Light Up, cells are either white or black, but are fully determined by the puzzle description. Therefore, choice formulas do not appear in the definition of either black or white cells.

```
cell(b,X,Y) :- black(X,Y,N).
cell(b,X,Y) :- black(X,Y).

cell(w,X,Y) :- not cell(b,X,Y), row(X), col(Y).
```

As the group of cells containing a lightbulb is a subset of the group containing all white cells, the placement of lightbulbs can be defined using a choice formula:

```
{lightbulb(X,Y)} :- cell(w,X,Y), row(X), row(Y).
```

As lightbulbs must be placed adjacent to numbered black cells, the concept of adjacency must be defined:

```
%define orthogonal adjacency
adj(X,Y,R,S) :- row(X;R), col(Y;S), abs(X-R)+abs(Y-S) == 1.
```

Adjacency is defined to apply only to orthogonal cells. According to the rule set, lightbulbs placed diagonally adjacent to a numbered black cell do not contribute to the bulb count.

Each numbered black cell represents the number of lightbulbs that should be placed adjacent to its four sides. Formalization of this rule requires a two-step process. First, a variable is introduced to record the placement of all bulbs placed adjacent to numbered black cells. A bulb is adjacent to a numbered black cell if and only if it is in an orthogonally adjacent cell to the numbered black cell. The following rule defines the set of adjacent bulbs:

```
%find bulbs adjacent to black numbered cells
adjbulb(X,Y,R,S) :- row(R), col(S), black(X,Y,N), lightbulb(R,S), adj(X,Y,R,S).
```

Second, in order to ensure that each numbered black cell has the appropriate bulb count, a new variable is introduced which defines adjacent bulb placement using a choice formula in conjunction with a cardinality constraint. In conjunction with this rule, a constraint is also included such that all black numbered cells are contained in the answer set:

```
%make sure each black numbered cell has same number of adjacent light bulbs
adj_bulb_placement(X,Y) :- N{adjbulb(X,Y,R,S):row(R):col(S)}N, black(X,Y,N).
:- not adj_bulb_placement(X,Y), black(X,Y,N).
```

Lightbulbs are placed on the grid such that no two lightbulbs shine on each other. Two lightbulbs shine on each other if and only if they are contained in the same row or column and no black cell is located between them. Therefore in order to formalize this rule, first it is necessary to introduce a new variable, which defines the concept of connectedness. Connectedness is defined recursively such that each cell is connected to itself and to each orthogonally adjacent white cell. Connectedness is defined by the following rules:

```
%define connectedness (only care about white cells)
connect_cell(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
connect_cell(X,Y,R,S) :- connect_cell(X,Y,T,U), adj(R,S,T,U),or(X==R,Y==S),
                                    cell(w,R,S), row(X;R;T), col(Y;S;U).
```

Second, in order to ensure that two bulbs do not shine on each other, a constraint is added such that if an answer set contains two lightbulbs such that each lightbulb is connected to the other that answer set violates the constraint and is removed. As each lightbulb is trivially connected to itself, the constraint must be defined such that it does

not apply to self-connected cells. If it did, every answer set would violate this constraint as every white cell is connected to itself and every lightbulb is a white cell. The inclusion of a disjunctive statement within the constraint such that either X or Y coordinates of the respective lightbulbs must be dissimilar ensures that self-connectedness does not violate the constraint, and therefore such cases are not excluded from the set of possible answer sets. The constraint is represented by the following rule:

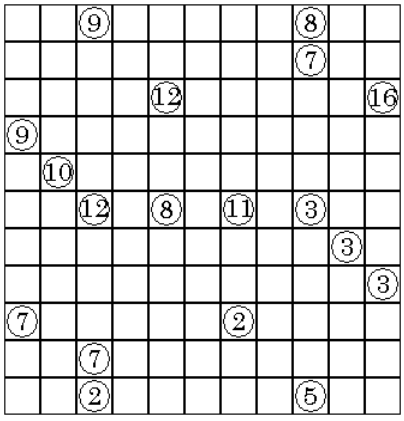
```
%make sure 2 lightbulbs are not connected
:- lightbulb(X,Y), lightbulb(R,S), connect_cell(X,Y,R,S), or(X!=R,Y!=S), row(X;R),
                        col(Y;S).
```

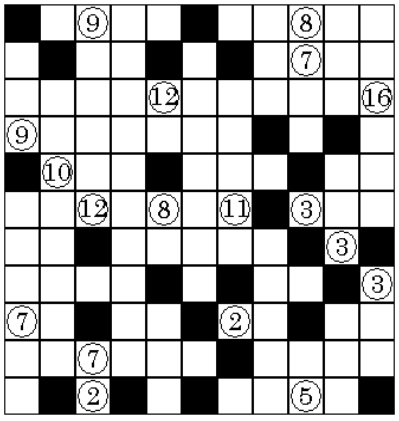
Every white cell contained in the grid must be lit by at least a single light bulb. A white cell is lit by a lightbulb if and only if it is connected to it. Therefore in order to ensure that all white cells are lit, a new variable is defined such that a white cell is lit if and only if it is connected to a lightbulb. A constraint is also added such that if an answer set contains a white cell that is not lit that answer set is removed from the set of possible answer sets. The constraint is represented by the following two rules:

```
%every white cell should be lit
lit(X,Y) :- lightbulb(R,S), connect_cell(X,Y,R,S), row(X;R), col(Y;S).
:- not lit(X,Y), cell(w,X,Y), row(X), col(Y).
```

Sample Puzzles

Kuromasu





Input:

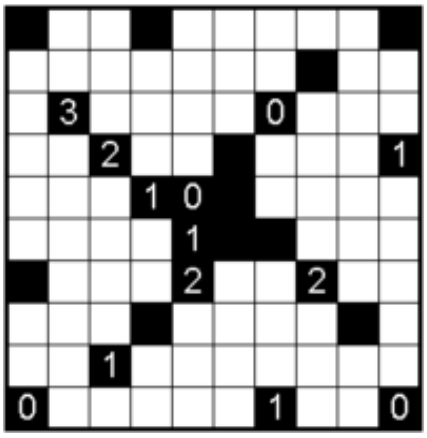
num(1,3,2). num(6,9,3). num(1,9,5). num(7,2,10).
num(2,3,7). num(8,1,9). num(3,1,7). num(9,5,12).
num(3,7,2). num(9,11,16). num(4,11,3). num(10,9,7).
num(5,10,3). num(11,3,9). num(6,3,12). num(11,9,8).
num(6,5,8). num(6,7,11).

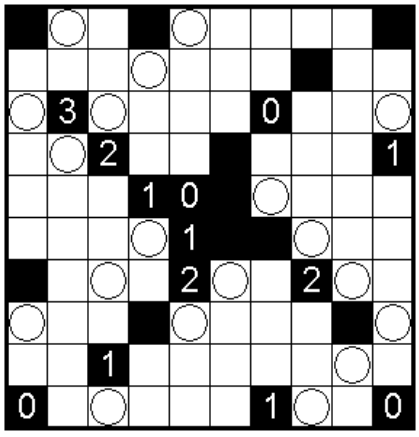
Output:

black(7,1) black(11,1) black(1,2) black(10,2) black(3,3) black(5,3) black(1,4) black(4,5) black(7,5) black(10,5) black(1,6) black(3,6) black(11,6) black(2,7) black(4,7) black(10,7) black(6,8) black(8,8) black(3,9) black(5,9) black(7,9) black(4,10) black(8,10) black(1,11) black(5,11)

Duration: 5.443
Number of atoms: 23298
Number of rules: 79738
Number of truth assignments: 526852

Light Up





Input:

black(1,7,1). black(3,9). black(1,10,0). black(4,1).
black(2,3,1). black(5,6). black(4,5,2). black(5,7).
black(4,8,2). black(6,6). black(5,5,1). black(7,6).
black(6,4,1). black(9,8). black(6,5,0). black(10,1).

black(7,3,2). black(10,4). black(7,10,1). black(10,10).
black(8,2,3). black(8,7,0).

Output:
lightbulb(3,1) lightbulb(8,1) lightbulb(7,2) lightbulb(10,2) lightbulb(1,3) lightbulb(4,3) lightbulb(8,3) lightbulb(5,4) lightbulb(9,4) lightbulb(3,5) lightbulb(10,5) lightbulb(4,6) lightbulb(6,7) lightbulb(1,8) lightbulb(5,8) lightbulb(2,9) lightbulb(4,9) lightbulb(3,10) lightbulb(8,10)

Duration: 0.825
Number of atoms: 12146
Number of rules: 27668
Number of truth assignments: 2724

CPU times are recorded in seconds and all experimental results were performed on a CPU with a 2.4 GHz Intel Core Quad processor and 2 X 2 GB SDRAM, running Windows Vista Business.

Conclusion

Both solvers provide a unique solution to each type of puzzle. However, the execution time for the Kuromasu solver is much slower than that of the Light Up solver. A possible explanation for the difference in execution time, first discovered while designing the Kuromasu solver, is associated with the implementation of the following rule: all white cells must be connected horizontally or vertically. This rule is implemented by the following:

```
white_connect(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
white_connect(X,Y,R,S) :- white_connect(X,Y,T,U), adj(R,S,T,U), cell(w,R,S),
                                    row(X;R;T), col(Y;S;U).
:- not white_connect(X,Y,R,S), cell(w,X,Y), cell(w,R,S), row(X;R), col(Y;S).
```

As this was the last rule implemented within the solver, the solver was tested both before and after inclusion. Before inclusion, the execution time was:

Duration: 0.935
Number of atoms: 8657
Number of rules: 10736

Number of truth assignments: 38998

And after inclusion:

Duration: 5.443
Number of atoms: 23298
Number of rules: 79738
Number of truth assignments: 526852

The number of atoms, rules, and truth assignments become much larger with the inclusion of this rule, and associated with higher numbers for these values is an increase in execution time. It is therefore apparent that the cost of including this rule within the solver is high. While this explanation is only cursory, future work, specifically on the relationship, if any, between problem specification and execution complexity is required in order to form a more robust conclusion on the matter.

Appendix

Kuromasu.lp

```
row(1..11).
col(1..11).
color(b;w).

num(1,3,2).        num(6,9,3).
num(1,9,5).        num(7,2,10).
num(2,3,7).        num(8,1,9).
num(3,1,7).        num(9,5,12).
num(3,7,2).        num(9,11,16).
num(4,11,3).       num(10,9,7).
num(5,10,3).       num(11,3,9).
num(6,3,12).       num(11,9,8).
num(6,5,8).        num(6,7,11).

%generate white cells
{cell(w,X,Y)} :- row(X), col(Y).

%numbered cells must be white
:- not cell(w,X,Y), num(X,Y,N).

%define black cells
cell(b,X,Y) :- not cell(w,X,Y), row(X), col(Y).

%define adjacency
adj(X,Y,R,S) :- row(X;R), col(Y;S), abs(X-R)+abs(Y-S) == 1.

%define connectedness for numbered cells (only white cells are numbered)
connect_cell(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
connect_cell(X,Y,R,S) :- connect_cell(X,Y,T,U), adj(R,S,T,U),or(X==R,Y==S), cell(w,R,S),
                                                    row(X;R;T), col(Y;S;U).

%define connectedness for white cells
white_connect(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
```

```
white_connect(X,Y,R,S) :- white_connect(X,Y,T,U), adj(R,S,T,U), cell(w,R,S), row(X;R;T),
col(Y;S;U).

%make sure every white cell is reachable from any given white cell
:- not white_connect(X,Y,R,S), cell(w,X,Y), cell(w,R,S), row(X;R), col(Y;S).

%every numbered cell should be seen by N white cells
seen(X,Y) :- N{connect_cell(X,Y,R,S):row(R):col(S)}N, num(X,Y,N).
:- not seen(X,Y), num(X,Y,N).

%no two black cells can be horizontally or vertically adjacent
:- cell(b,X,Y), cell(b,R,S), adj(X,Y,R,S), row(X;R), col(Y;S).

black(X,Y) :- cell(b,X,Y), row(X), col(Y).

hide.
show black(_,_).
```

lightup.lp

```
row(1..10).
col(1..10).
color(b;w).

black(1,1,0).
black(1,7,1).
black(1,10,0).
black(2,3,1).
black(4,5,2).
black(4,8,2).
black(5,5,1).
black(6,4,1).
black(6,5,0).
black(7,3,2).
black(7,10,1).
black(8,2,3).
black(8,7,0).
black(3,4).
black(3,9).
black(4,1).
black(5,6).
black(5,7).
black(6,6).
black(7,6).
black(9,8).
black(10,1).
black(10,4).
black(10,10).

%designate black cells
cell(b,X,Y) :- black(X,Y,N).
cell(b,X,Y) :- black(X,Y).

%if cell is not black, it is white
cell(w,X,Y) :- not cell(b,X,Y), row(X), col(Y).

%define orthogonal adjacency
adj(X,Y,R,S) :- row(X;R), col(Y;S), abs(X-R)+abs(Y-S) == 1.

%define connectedness (only care about white cells)
connect_cell(X,Y,X,Y) :- cell(w,X,Y), row(X), col(Y).
connect_cell(X,Y,R,S) :- connect_cell(X,Y,T,U), adj(R,S,T,U),or(X==R,Y==S), cell(w,R,S),
row(X;R;T), col(Y;S;U).

%generate lightbulbs (subset of white cells)
{lightbulb(X,Y)} :- cell(w,X,Y), row(X), col(Y).
```

```
%make sure 2 lightbulbs are not connected
:- lightbulb(X,Y), lightbulb(R,S), connect_cell(X,Y,R,S), or(X!=R,Y!=S), row(X;R), col(Y;S).

%every white cell should be lit
lit(X,Y) :- lightbulb(R,S), connect_cell(X,Y,R,S), row(X;R), col(Y;S).
:- not lit(X,Y), cell(w,X,Y), row(X), col(Y).

%find bulbs adjacent to black numbered cells
adjbulb(X,Y,R,S) :- row(R), col(S), black(X,Y,N), lightbulb(R,S), adj(X,Y,R,S).

%make sure each black numbered cell has same number of adjacent light bulbs
adj_bulb_placement(X,Y) :- N{adjbulb(X,Y,R,S):row(R):col(S)}N, black(X,Y,N).
:- not adj_bulb_placement(X,Y), black(X,Y,N).

hide.
show lightbulb(_,_).
```