COMP4418 – Assignment 2

Submitted by z5222191.

Part 1

Given a graph (V, E) an independent set is a set of vertices I ⊆ V such that no pair of vertices in I is adjacent. Develop an ASP program that outputs an independent set of size given as parameter. Use the unary predicate select/1 to indicate which set of vertices should be selected in an independent set.

Output for the given sample test cases :

```
bash-3.2$ clingo --const size=4 independent.lp petersen.lp
clingo version 5.4.0
Reading from independent.lp ...
Solving...
Answer: 1
select(1) select(3) select(10) select(9)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
bash-3.2$
```

```
bash-3.2$ clingo --const size=5 independent.lp petersen.lp
clingo version 5.4.0
Reading from independent.lp ...
Solving...
UNSATISFIABLE

Models       : 0
Calls        : 1
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
bash-3.2$
```

Part 2

Given a direct graph (V, E), a feedback edge set is a set of edges F ⊆ E such that removing these edges leaves an acyclic subgraph. That is, F is a feedback edge set iff (V, E \ F) is a directed acyclic graph. Develop an ASP program that outputs a feedback edge set of size given as parameter. Use the binary predicate select/2 to indicate which set of edges should be selected in a feedback edge set.

Output for the given sample test cases :

```
bash-3.2$ clingo --const size=15 feedback.lp petersen.lp
clingo version 5.4.0
Reading from feedback.lp ...
Solving...
Answer: 1
select(2,3) select(3,4) select(4,5) select(5,1) select(10,7) select(7,9) select(1,6) select(2,7) select(3,8) select(4,9) select(10,5) select(6,9) select(10,8) select(
8,6) select(2,1)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.004s
```

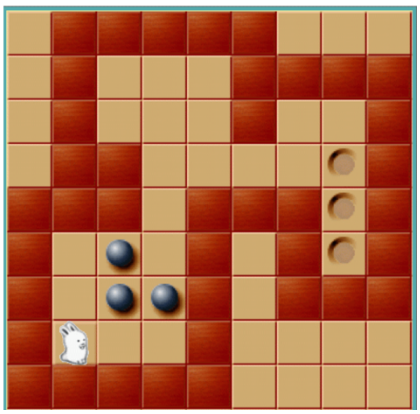
```
bash-3.2$ clingo --const size=14 feedback.lp petersen.lp
clingo version 5.4.0
Reading from feedback.lp ...
Solving...
UNSATISFIABLE

Models       : 0
Calls        : 1
Time         : 0.024s (Solving: 0.02s 1st Model: 0.00s Unsat: 0.02s)
CPU Time     : 0.024s
bash-3.2$
```

The total number of models for the problem statement with size = 14 is given below.

```
Models       : 16680
Calls        : 1
Time         : 0.494s (Solving: 0.49s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.322s
```

Part 3

The given Sokoban board can be represented as a two dimensional grid with coordinates as given in the matrix on the right.



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

In the given Sokoban board, the coordinates of the player P is (2,2) and the coordinates of the boulders B1, B2, and B3 are (3,4), (3,3) and (4,3) respectively.

Question 3.1 List the fixed and the dynamic relations.

The dynamic relations are :
emptyspace(i, j).
position(X,i,j).

The static relations are :
wall(i,j).
adjacent_to_goal(i,j).

The operators are:
move(i, j, k, l).

push(i, j, k, l, m, n).
The operators ‘move’ and ‘push’ are dynamic in nature.

Question 3.2 Define the preconditions and effects of each operator.
The first operator involves moving the player from given position to new position.
Generally, for any move from i,j to k,l the space k,l and every space adjacent to k,l in the direction of traversal must be empty.
General case :
Preconditions for move(i,j,k,l):
Position(P,i,j). emptyspace(k,l).
Positive Effects : position(P,k,l), emptyspace(i,j).
Negative Effects : not position(P,i,j), not emptyspace(k,l).

Preconditions for push(i,j,k,l,m,n,o,p) :
Position(P,i,j). position(B,m,n). emptyspace(k,l). emptyspace(o,p).
Positive Effects : position(P,k,l), position(B,o,p), emptyspace(i,j). emptyspace(m,n).
Negative Effects : not position(P,i,j). not position(B,m,n). not emptyspace(k,l). not emptyspace(o,p).

Scenario : Moving B3 from initial state to goal state G3.
Preconditions for move(2,2,4,2) :
emptyspace(3,2). emptyspace(4,2). not wall(3,2), not wall(4,2).
Action : move(2,2,4,2).
Positive Effects : emptyspace(2,2), position(P,4,2).
Negative Effects : not emptyspace(4,2). Not position(P,2,2).

Preconditions for push(4,2,4,6,4,3,4,7) :
emptyspace(4,3). emptyspace(4,4). emptyspace(4,5). emptyspace(4,6). emptyspace(4,7).
not wall(4,8). not wall(5,7). emptyspace(4,8). emptyspace(5,7). position(P,4,2). position(B3, 4,3).
Action : push(4,2,4,6,4,3,4,7).
Positive Effects : position(P,4,6). position(B3, 4,7). emptyspace(4,2). emptyspace(4,3).
Negative Effects : not empty(4,6). not empty(4,7).

Preconditions for move(4,6,5,6) :
emptyspace(5,6).
Action : move(4,6,5,6)**.**
Positive Effects : position(P,5,6), emptyspace(4,6).
Negative Effects : not emptyspace(5,6).

Preconditions for move(5,6,5,8) :
emptyspace(5,7). emptyspace(5,8).
Action : move(5,6,5,8)**.**
Positive Effects : position(P,5,8), emptyspace(5,6).
Negative Effects : not emptyspace(5,8).

Preconditions for move(5,8,3,8) :
emptyspace(4,8). emptyspace(3,8).
Action : move(5,8,3,8)**.**
Positive Effects : position(P,3,8), emptyspace(5,8).
Negative Effects : not emptyspace(3,8).

Preconditions for move(3,8,3,7) :
emptyspace(3,7).
Action : move(3,8,3,7)**.**
Positive Effects : position(P,3,7). emptyspace(3,8).

Negative Effects : not emptyspace(3,7).

Preconditions for push(3,7,4,7,4,7,5,7) :.
not wall(5,8). not wall(5,7). emptyspace(5,7). position(P,3,7). position(B3, 4,7).
Action : push(4,2,4,6,4,3,4,7).
Positive Effects : position(P,4,7). position(B3, 5,7). emptyspace(3,7). emptyspace(4,7).
Negative Effects : not empty(4,7). not empty(5,7).

Preconditions for move(4,7,4,8) :
emptyspace(4,8).
Action : move(4,7,4,8)**.**
Positive Effects : position(P,4,8). emptyspace(4,7).
Negative Effects : not emptyspace(4,8).

Preconditions for move(4,8,5,8) :
emptyspace(5,8).
Action : move(4,8,5,8)**.**
Positive Effects : position(P,5,8). emptyspace(4,8).
Negative Effects : not emptyspace(5,8).

Preconditions for push(5,8,5,7,5,7,5,6) :
not wall(5,7). not wall(5,6). position(P,5,8). position(B3, 5,7).
Action : push(5,8,5,7,5,7,5,6)
Positive Effects : position(P,5,7). position(B3, 5,6). emptyspace(5,8).
Negative Effects : not empty(5,7). not empty(5,6).

Preconditions for move(5,7,4,7) :
emptyspace(4,7).
Action : move(5,7,4,7)**.**
Positive Effects : position(P,4,7). emptyspace(5,7).
Negative Effects : not emptyspace(4,7).

Preconditions for move(4,7,4,6) :
emptyspace(4,6).
Action : move(4,7,4,6)**.**
Positive Effects : position(P,4,6). emptyspace(4,7).
Negative Effects : not emptyspace(4,6).

Preconditions for push(4,6,7,6,5,6,8,6) :
not wall(7,6). not wall(8,6). position(P,7,6). position(B3, 8,6). emptyspace(7,6). emptyspace(8,6).
Action : push(4,6,7,6,5,6,8,6)
Positive Effects : position(P,7,6). position(B3, 8,6). emptyspace(4,6). emptyspace(5,6).
Negative Effects : not empty(7,6). not empty(8,6).

Preconditions for move(7,6,7,7) :
emptyspace(7,7).
Action : move(7,6,7,7).
Positive Effects : position(P,7,7). emptyspace(7,6).
Negative Effects : not emptyspace(7,7).

Preconditions for move(7,7,8,7) :
emptyspace(8,7).
Action : move(7,7,8,7).
Positive Effects : position(P,7,8). emptyspace(7,7).
Negative Effects : not emptyspace(8,7).

Preconditions for push(8,7,8,5,8,6,8,4) :
not wall(8,5). not wall(8,4). position(P,8,7). position(B3, 8,6). emptyspace(8,5). emptyspace(8,4).
Action : push(8,7,8,5,8,6,8,4).
Positive Effects : position(P,8,5). position(B3, 8,4). emptyspace(8,7). emptyspace(8,5). adjacenttogoal(P,8,4).
Negative Effects : not empty(8,5). not empty(8,4).

Goal state for B3 achieved.

Question 3.3 Define the initial state corresponding to instance in the figure. If some parts of the answer appear to be very repetitive, it is possible to use ellipsis ". . . " and not list everything. Use your best judgement.

The initial state of the Sokoban board corresponding to the instance in the figure can be defined as the following :
S0 = {
position(P,2,2).
position(B1,3,4).
position(B2,3,3).
position(B3,4,3).
position(G3,8,6)
position(G2,8,5)
position(G1,8,4)
(Here, P refers to the player.
B1, B2, and B3 refer to the respective boulders at their positions.
G1, G2, G3 refer to the respective goals at their positions.)
wall(2,9), wall(3,9), wall(4,9), wall(5,9), wall(6,9).
wall(2,8), wall(6,8), wall(7,8), wall(8,8), wall(9,8).
wall(2,7), wall(6,7), wall(9,7).
wall(2,6), wall(3,6), wall(9,6).
wall(1,5), wall(2,5), wall(3,5), wall(5,5), wall(6,5), wall(7,5), wall(9,5).
wall(1,4), wall(5,4), wall(7,4), wall(9,4).
wall(1,3), wall(5,3), wall(7,3), wall(8,3), wall(9,3).
wall(1,2), wall(5,2).
wall(1,1), wall(2,1), wall(3,1), wall(4,1), wall(5,1).
emptyspace(1,9), emptyspace(7,9), emptyspace(8,9), emptyspace(9,9).
emptyspace(1,8), emptyspace(3,8), emptyspace(4,8), emptyspace(5,8).
emptyspace(1,7),emptyspace(3,7),emptyspace(4,7),emptyspace(5,7),emptyspace(7,7),
emptyspace(8,7).
emptyspace(1,6),emptyspace(4,6),emptyspace(5,6),emptyspace(6,6),emptyspace(7,6),
emptyspace(8,6).
emptyspace(4,5), emptyspace(8,5).
emptyspace(2,4), emptyspace(4,4), emptyspace(6,4), emptyspace(8,4).
emptyspace(2,3), emptyspace(6,3).
emptyspace(3,2),emptyspace(4,2),emptyspace(6,2),emptyspace(7,2),emptyspace(8,2),
emptyspace(9,2).
emptyspace(6,1), emptyspace(7,1), emptyspace(8,1), emptyspace(9,1).
}

Question 3.4 Define the goal corresponding to the instance in the figure.

The goal is achieved when the positions of the boulders are the positions of the goals.
Goal = { position(B1,8,6), position(B2,8,5), position(B3,8,4). position(P,7,6).}

The Sokoban board at the goal state will have a model that can be represented as
Sg = {
wall(2,9), wall(3,9), wall(4,9), wall(5,9), wall(6,9).
wall(2,8), wall(6,8), wall(7,8), wall(8,8), wall(9,8).
wall(2,7), wall(6,7), wall(9,7).
wall(2,6), wall(3,6), wall(9,6).
wall(1,5), wall(2,5), wall(3,5), wall(5,5), wall(6,5), wall(7,5), wall(9,5).
wall(1,4), wall(5,4), wall(7,4), wall(9,4).
wall(1,3), wall(5,3), wall(7,3), wall(8,3), wall(9,3).
wall(1,2), wall(5,2).
wall(1,1), wall(2,1), wall(3,1), wall(4,1), wall(5,1).
emptyspace(1,9), emptyspace(7,9), emptyspace(8,9), emptyspace(9,9).
emptyspace(1,8), emptyspace(3,8), emptyspace(5,8).
emptyspace(1,7),emptyspace(3,7),emptyspace(4,7),emptyspace(5,7),emptyspace(7,7),
emptyspace(8,7).
emptyspace(1,6),emptyspace(4,6),emptyspace(5,6),emptyspace(6,6).
emptyspace(4,5).
emptyspace(2,4), emptyspace(4,4), emptyspace(6,4), emptyspace(8,4).
emptyspace(2,3), emptyspace(6,3).
emptyspace(3,2),emptyspace(4,2),emptyspace(6,2),emptyspace(7,2),emptyspace(8,2),
emptyspace(9,2).
emptyspace(6,1), emptyspace(7,1), emptyspace(8,1), emptyspace(9,1).
emptyspace(2,2), emptyspace(3,3), emptyspace(4,3), emptyspace(3,4).
position(B1,8,6), position(B2,8,5), position(B3,8,4). position(P,7,6).
}

Question 3.5 What is the main obstacle to using Answer Set Programming to solve Sokoban instances? (Answer with no more than one or two sentences in English.)

ASP or any SAT solver is designed to check if there are any stable models or solutions to a given problem statement that cannot be solved in polynomial complexity time or in other words ASP is specifically designed to solve NP- complete problems. The Sokoban puzzle is a PSPACE problem and though there could be a solution to a PSPACE problem, there is no easy way to verify whether it satisfies a given rule. ASP looks for solutions by expressing an instance of the problem through rules and conditions. Then, looks for solutions that satisfy the above-mentioned rules and conditions. Sokoban, however cannot be expressed in such instances.