Universidad Simón Bolívar

Inteligencia Artificial I: Proyecto 3

Para el último proyecto de la asignatura Inteligencia Artificial I (CI-5437), se solicitó construir un programa que solucione instancias arbitrarias del juego descrito en el enunciado, utilizando un SAT solver. Las instancias fueron provistas en un archivo *input.txt.*

Representación del problema.

Antes de construir el programa requerido, fue necesario generar las cláusulas que restringen la solución, para ello se utilizó la representación dada como ayuda para el proyecto, con algunos cambios puntuales, de la siguiente manera:

Una celda es representada como c(i,j) con etiqueta n, donde:
i = número de filas del tablero.
j = número de columnas del tablero.

De esta manera, resultó mucho más intuitiva la generación del programa, por lo que se decidió adaptar las cláusulas a este modelo.

A continuación se listan las cláusulas generadas escritas en lógica proposicional y su representación en CNF. Se implementaron 5 categorías de cláusulas:

- Tipo 0: Las cláusulas de tipo 0 nos indican que para cada celda, si esta posee un segmento al este, la celda a su derecha lo posee a su izquierda. Al igual para el caso del norte y el sur.

  Lógica proposicional:

  ∀ (i,j) ((1 <= i < N && 1 <= j <= M) => (-q(i,j,e) <=> -q(i+1,j,w)))

  ∀ (i,j) ((1 <= i <= N && 1 <= j < M) => (-q(i,j,n) <=> -q(i,j+1,s)))

  CNF:

  ∀ (i,j) ((1 <= i < N && 1 <= j <= M) => (q(i,j,e) || -q(i+1,j,w))
  && (-q(i,j,e) || q(i+1,j,w)))

  ∀ (i,j) ((1 <= i <= N && 1 <= j < M) => (q(i,j,n) || -q(i,j+1,s))
  && (-q(i,j,n) || q(i,j+1,s)))

- Tipo 1: Las cláusulas de tipo 1 se encargan de forzar que para todas las celdas, se tenga el número correcto (0, 1, 2, 3 o 4) de segmentos a su alrededor en la solución.

n = 0
Lógica proposicional:

∀ (i,j) ((1 <= i < N && 1 <= j <= M) => (-q(i,j,n) && -q(i,j,s)
&& -q(i,j,e) && -q(i,j,w)))

CNF: Misma representación.

n = 1
Lógica proposicional:

∀ (i,j) ((1 <= i < N && 1 <= j <= M) =>
((q(i,j,n) || q(i,j,s) || q(i,j,e) || q(i,j,w)) &&
(-q(i,j,n) || -q(i,j,s)) && (-q(i,j,n) || -q(i,j,e)) &&
(-q(i,j,n) || -q(i,j,w)) && (-q(i,j,e) || -q(i,j,s)) &&
(-q(i,j,e) || -q(i,j,w)) && (-q(i,j,s) || -q(i,j,w))))

CNF: Misma representación

n=2
Lógica proposicional:

∀ (i,j) ((1 <= i < N && 1 <= j <= M) =>
((-q(i,j,n) && -q(i,j,e) => q(i,j,s) && q(i,j,w)) &&
(-q(i,j,n) && -q(i,j,s) => q(i,j,e) && q(i,j,w)) &&
(-q(i,j,n) && -q(i,j,w) => q(i,j,s) && q(i,j,e)) &&
(-q(i,j,e) && -q(i,j,s) => q(i,j,n) && q(i,j,w)) &&
(-q(i,j,e) && -q(i,j,w) => q(i,j,n) && q(i,j,s)) &&
(-q(i,j,s) && -q(i,j,w) => q(i,j,n) && q(i,j,e))))

CNF:

∀ (i,j) ((1 <= i < N && 1 <= j <= M) => ((q(i,j,n) || q(i,j,e) || q(i,j,s))
&& q(i,j,n) || q(i,j,e) || q(i,j,w))
&& (q(i,j,e) || q(i,j,s) || q(i,j,w))
&& (q(i,j,n) || q(i,j,s) || q(i,j,w))))

n = 3
Lógica proposicional:

```
∀ (i,j) ((1 <= i < N && 1 <= j <= M) =>
        ( (-q(i,j,n) => q(i,j,s) && q(i,j,e) && q(i,j,w))
        && (-q(i,j,s) => q(i,j,n) && q(i,j,e) && q(i,j,w))
        && (-q(i,j,e) => q(i,j,n) && q(i,j,s) && q(i,j,w))
        && (-q(i,j,w) => q(i,j,n) && q(i,j,s) && q(i,j,e)) ))
```

CNF:

```
∀ (i,j) ((1 <= i < N && 1 <= j <= M) => ((-q(i,j,n) || -q(i,j,s) || -q(i,j,e) || -q(i,j,w))))
```

n = 4
Lógica proposicional:

```
∀ (i,j) ((1 <= i < N && 1 <= j <= M) =>
        ((q(i,j,n) && q(i,j,s) && q(i,j,e) && q(i,j,w))))
```

CNF: Misma representación.

- Tipo 2: Las cláusulas de tipo 2 son las que se encargan de indicarnos para todas las celdas si esta se encuentra dentro o fuera del perímetro.

n=4
Lógica proposicional:

```
∀ (i,j) ((1 <= i <= N && 1 <= j <= M) =>
        q(i,j,n) && q(i,j,s) && q(i,j,e) && q(i,j,w) => -z(i,j) )
```

CNF:

```
∀ (i,j) ((1 <= i <= N && 1 <= j <= M) =>
        ((-q(i,j,n) || -q(i,j,s) || -q(i,j,e) || -q(i,j,w) || -z(i,j))))
```

i=1
Lógica proposicional:

```
∀  ((1 <= j <= M) => -q(1,j,n) <=> z(1,j))
```

CNF:

∀ (j) ((1 <= j <= M) => ((q(1,j,n) || z(1,j))) && (-q(1,j,n) || -z(1,j))))

i=N

Lógica proposicional:

∀ (j) ((1 <= j <= M) => -q(N,j,s) <=> z(N,j))

CNF:

∀ (j) ((1 <= j <= M) => ((q(N,j,s) || z(N,j)) &&(-q(N,j,n) || -z(N,j))))

j=1

Lógica proposicional:

∀ (i) ((1 <= i <= N) => -q(i,1,w) <=> z(i,1))

CNF:

∀ (i) ((1 <= i <= N) => ((q(i,1,w) || z(i,1)) && (-q(i,1,w) || -z(i,1))))

j=M

Lógica proposicional:

∀ (i) ((1 <= i <= N) => -q(i,M,e) <=> z(i,M))

CNF:

∀ (i) ((1 <= i <= N) => ((q(i,M,e) || z(i,M)) && (-q(i,M,e) || -z(i,M))))

celdas internas

Lógica proposicional:

∀ (i,j) (1 < i < N && 1 < j < M) =>
z(i,j) <=> [-q(i,j,n) & z(i-1,j)] || [-q(i,j,s) & z(i+1,j)]
|| [-q(i,j,e) & z(i,j+1)] || [-q(i,j,w) & z(i,j-1)])

CNF:

```
∀ (i,j) (1 < i < N && 1 < j < M) =>
        (-q(i,j,e) || -q(i,j,w)) &&  (z(i,j+1) || -q(i,j,w)) &&
        (-q(i,j,e) || z(i,j-1)) &&  (z(i,j+1) || z(i,j-1)) &&
        (-z(i,j) || -q(i,j,n) || -q(i,j,s)) &&  (-z(i,j) || z(i-1,j) || -q(i,j,s)) &&
        (-z(i,j) || -q(i,j,n) || z(i+1,j)) &&  (-z(i,j) || z(i-1,j) || z(i+1,j)) &&
        z(i,j) || q(i,j,n) || -z(i-1,j)) &&  (z(i,j) || q(i,j,s) || -z(i+1,j)) &&
        (z(i,j) || q(i,j,e) || -z(i,j+1)) && (z(i,j) || q(i,j,w) || -z(i,j-1))&&
         (-z(i,j) || z(i+1,j)) && (-z(i,j) || -q(i,j,e)) &&  (-z(i,j) || z(i,j+1))
        && (-z(i,j) || -q(i,j,w)) && (-z(i,j) || z(i,j-1)) ) )
```

- Tipo 3: Las cláusulas de tipo 3 se encargan de definir cuando una celda es alcanzable desde otra. Es una definición inductiva que tradujimos de la siguiente manera:

  Cuando la de arriba es alcanzable
  Lógica proposicional:

```
∀ (i,j) ((1 <= i <= N && 1 <= j <= M) =>
        existe(i',j') ((1 < i' <= N && 1 <= j' <= M) =>
                c(i'-1,j') && r(c(i,j),c(i',j')) && -q(i',j',n) => r(c(i',j'),c(i'-1,j'))))
```

  CNF:

```
∀ (c(i,j)) ((1 <= i <= N && 1 <= j <= M) =>
        ( -r(c,c') || q(c',n) || r(c, c''))
```

  Cuando la de la derecha es alcanzable
  Lógica proposicional:

```
∀ (i,j) ((1 <= i <= N && 1 <= j <= M) =>
        existe(i',j') ((1 <= i' <= N && 1 <= j' < M) =>
                c(i',j'+1) && r(c(i,j),c(i',j')) && -q(i',j',e) =>
                        r(c(i',j'),c(i',j'+1))))
```

  CNF:

```
∀ (c(i,j)) ((1 <= i <= N && 1 <= j <= M) =>
        (-r(c,c') || q(c',e) || r(c, c'') )
```

  Cuando la de abajo es alcanzable

Lógica proposicional:

```
∀ (i,j) ((1 <= i <= N && 1 <= j <= M) =>
        existe(i',j') ((1 <= i' < N && 1 <= j' <= M) =>
                c(i'+1,j') && r(c(i,j),c(i',j')) && -q(i',j',s) =>
                r(c(i',j'),c(i'+1,j'))))
```

CNF:

```
∀ (c(i,j)) ((1 <= i <= N && 1 <= j <= M) =>
        (-r(c,c') || q(c',s) || r(c, c'') )
```

Cuando la de la izquierda es alcanzable

Lógica proposicional:

```
∀ (i,j) ((1 <= i <= N && 1 <= j <= M) =>
        existe(i',j') ((1 <= i' <= N && 1 < j' <= M) =>
                c(i',j'-1) && r(c(i,j),c(i',j')) && -q(i',j',w) =>
                        r(c(i',j'),c(i',j'-1))))
```

CNF:

```
∀ (c(i,j)) ((1 <= i <= N && 1 <= j <= M) =>
        (-r(c,c') || q(c',o) || r(c, c'') )
```

- Tipo 4: Las cláusulas de tipo 4 nos permiten determinar para cada par de celdas que están en el perímetro, estas son adyacentes.

  Lógica proposicional:

```
∀ (i,j,i',j') ((1 <= i <= N && 1 <= j <= M) && (1 <= i' <= N && 1 <= j' <= M) =>
        z(i,j) && z(i',j') => r(c(i,j),c(i',j'))
```

  CNF:

```
∀ (i,j,i',j') ((1 <= i <= N && 1 <= j <= M) && (1 <= i' <= N && 1 <= j' <= M) =>
        -z(i,j) || -z(i',j') || r(c(i,j),c(i',j')) )
```

- Tipo 5: Las cláusulas de tipo 5 nos indican si los puntos se encuentran bien conectados con cero o más incidencias.

  Esquina superior izquierda
  Lógica proposicional:

  (-q(1,1,n) && -q(1,1,w)) || (q(1,1,n) && q(1,1,w))

  CNF:

  (-q(1,1,n) || q(1,1,w)) && (q(1,1,n) || -q(1,1,w))

  Esquina superior derecha
  Lógica proposicional:

  (-q(1,M,n) && -q(1,M,e)) || (q(1,M,n) && q(1,M,e))

  CNF:

  (-q(1,M,n) || q(1,M,e)) && (q(1,M,n) || -q(1,M,e))

  Esquina inferior izquierda
  Lógica proposicional:

  (-q(N,1,s) && -q(N,1,w)) || (q(N,1,s) && q(N,1,w))

  CNF:

  (-q(N,1,s) || q(N,1,w)) && (q(N,1,s) || -q(N,1,w))

  Esquina inferior derecha
  Lógica proposicional:

  (-q(N,M,s) && -q(N,M,e)) || (q(N,M,s) && q(N,M,e))

  CNF:

  (-q(N,M,s) || q(N,M,e)) && (q(N,M,s) || -q(N,M,e))

  Lateral superior
  Lógica proposicional:

  ∀ (j) ((0 < j < M) => (-q(1,j,n) && -q(1,j,e) && -q(1,j+1,n)) ||
  ((q(1,j,e) && (q(1,j,n) || q(1,j+1,n))) ||(q(1,j,n) && q(1,j+1,n))))

CNF:

```
∀ (j) ((0 < j < M) => ((-q(1,j,n) || q(1,j,e) || q(1,j+1,n))
        &&(q(1,j,n) || -q(1,j,e) || q(1,j+1,n))
       &&(q(1,j,n) || q(1,j,e) || -q(1,j+1,n)) ) )
```

Lateral inferior

Lógica proposicional:

```
∀ (j) ((0 < j < M) => (-q(N,j,s) && -q(N,j,e) && -q(N,j+1,s)) ||
       ((q(N,j,e) && (q(N,j,s) || q(N,j+1,s))) ||
       (q(N,j,s) && q(N,j+1,s)))
```

CNF:

```
∀ (j) ((0 < j < M) => ((-q(N,j,s) || q(N,j,e) || q(N,j+1,s))
        &&(q(N,j,s) || -q(N,j,e) || q(N,j+1,s))
       &&(q(N,j,s) || q(N,j,e) || -q(N,j+1,s))))
```

Lateral izquierda

Lógica proposicional:

```
∀ (i) ((0 < i < N) => (-q(i,1,w) && -q(i,1,s) && -q(i+1,1,w))
        || ( (q(i,1,s) && (q(i,1,w) || q(i+1,1,w)))
       || (q(i,1,w) && q(i+1,1,w)) )
```

CNF:

```
∀ (i) ((0 < i < N) =>  ( (-q(i,1,w) || q(i,1,s) || q(i+1,1,w))
       && (q(i,1,w) || -q(i,1,s) || q(i+1,1,w))
        && (q(i,1,w) || q(i,1,s) || -q(i+1,1,w))))
```

Lateral derecha

Lógica proposicional:

```
∀ (i) ((0 < i < N) =>  (-q(i,M,e) && -q(i,M,s) && -q(i+1,M,e))
        || ( (q(i,M,s) && (q(i,M,e) || q(i+1,M,e)))
       || (q(i,M,e) && q(i+1,M,e)) )
```

CNF:

∀ (i) ((0 < i < N) =>((-q(i,M,e) || q(i,M,s) || q(i+1,M,e))
&& (q(i,M,e) || -q(i,M,s) || q(i+1,M,e))
&& (q(i,M,e) || q(i,M,s) || -q(i+1,M,e))))

Centrales
Lógica proposicional:

∀ (i,j) ((0 < i < N && 0 < j < M) =>
(-q(i,j,s) && -q(i,j,e) && -q(i+1,j+1,n) && -q(i+1,j+1,w))
|| ((q(i,j,s) && q(i,j,e)) || (q(i,j,s) && q(i+1,j+1,n))
|| (q(i,j,s) && q(i+1,j+1,w)) || (q(i,j,e) && q(i+1,j+1,n))
|| (q(i,j,e) && q(i+1,j+1,w)) || (q(i+1,j+1,n) && q(i+1,j+1,w)))

CNF:

∀ (i,j) ((0 < i < N && 0 < j < M) =>
((-q(i,j,s) || q(i,j,e) || q(i+1,j+1,n) || q(i+1,j+1,w))
&& (q(i,j,s) || -q(i,j,e) || q(i+1,j+1,n) || q(i+1,j+1,w))
&& (q(i,j,s) || q(i,j,e) || -q(i+1,j+1,n) || q(i+1,j+1,w))
&& (q(i,j,s) || q(i,j,e) || q(i+1,j+1,n) || -q(i+1,j+1,w)))

Corrida del programa.

Para la corrida del programa se requiere que la máquina posea el SAT solver *minisat* instalado, para ello:

sudo apt-get install minisat

Luego de tener instalado el solver, se ejecuta el script de corrida como sigue:

./solver.sh <input>.txt

Dónde “*input*” corresponde al nombre del archivo de texto donde se encuentran las instancias arbitrarias a resolver. Este script genera un archivo “*output.txt”* donde se encuentra la solución provista por el SAT solver luego de ser versionada a formato de salida requerido.

Resultados.

De la información obtenida en *output.txt* se observó que sólo una de las instancias del archivo *input.txt* resultó ser insatisfacible.