SAT Solver: MineSweeper

Pedro Rodriguez - 15-11264 and Daniel Pinto - 15-11139

1 Specs del equipo usado

Todas las pruebas se corrieron en el mismo equipo, el cual cuenta con los siguientes specs:

- **Procesador:** Ryzen 7 3800x, 3.9GHz.
- **RAM:** 16GB, 3200MHz.
- **Java Heap Size**: 4GB
- SSD.

2 Modelado de variables Inicial

Inicialmente pensamos en encontrar todas las posiciones en donde se puede jugar seguro, sin embargo, existen configuraciones en donde no todas las minas están completamente determinadas, sino parcialmente determinadas, como:

```
____2____
---------
---------
---------
```

En donde cualquier 2 combinación de minas es peligrosa. Por lo tanto, atacamos el problema dual: seleccionar cuales minas se pueden colocar con certeza. Por lo tanto, nuestra codificacion se reduce a:

```
type P = (Int,Int)
```

En donde $P_{i,j}$ representa que en la posición i, j de la matriz debe existir una mina.

3 Restricciones

Debido a la buena escogencia de la variable P, la generación de restricciones se trivializa al siguiente predicado:

Sea M el mapa del buscamina, $n \in N \subset \mathbb{N} \times \mathbb{N}$ una casilla que pertenece al conjunto de casillas numéricas, y sea $neigh : \mathbb{N} \times \mathbb{N} \to \mathscr{P}(\mathbb{N} \times \mathbb{N})$ una función que genera la vecindad de la posición N (sin contarse a ella misma), entonces definimos:

$$neighValid\ n = \{p \mid\ p \in neigh(n) \wedge M(p) = BLANK\}$$

Como el conjunto de posiciones validas en donde se puede posicionar una mina, y por lo tanto nuestras restricciones serán:

Para cada $n \in NN \subset \mathbb{N} \times \mathbb{N}$ con numero de minas n se debe de cumplir:

Pedro Rodriguez - 15-11264: 15-11264@usb.ve
Daniel Pinto - 15-11139: 15-11139@usb.ve

$$F_n = \bigvee \left\{ \bigwedge minas \wedge \bigwedge \sim noMinas \mid \ minas \in \binom{neighValid(n)}{N} \wedge noMinas = neigh(n) \backslash minas \right\}$$

Esto nos genera todas las posibles restricciones para las minas al rededor de un numero en especifico, para todos los números, basta con generalizar para todo n:

$$\bigwedge \{F_n \mid n \in N \subset \mathbb{N} \times \mathbb{N}\}$$

Notemos que al igual que el proyecto anterior, tenemos algo de la forma $\vee\wedge$, sin embargo, tenemos una base (cantidad de espacios disponibles) considerablemente menor puesto que a lo sumo se tienen 8 direcciones, y 8 combinaciones.

4 Parafernalia y Miscelánea

Esta sección tiene como objetivo describir la estructura del proyecto. En particular como correrlo.

El lenguaje seleccionado fue Scala 3.0 con el manejador de paquetes SBT. Por lo tanto, es altamente recomendable utilizar un IDE como intellij para compilar y correr el proyecto.

El proyecto utiliza el archivo de configuración `options.yaml` para configurar todas las variables que necesita el proyecto, en particular, posee el siguiente esquema:

```
entryPoint     : path/a/donde/esta/el/mapa/a/leer
glucoseExecutable : /path/al/ejecutable/de/glucose
glucoseInput   : /path/a/donde/se/guarda/el/input/de/glucose
glucoseOutput  : /path/a/donde/se/guarda/el/output/de/glucose
output         : /path/donde/se/guardan/los/resultados
```