Projet Logique
2014/15
Première partie

Stefan Schwoon

Introduction

The first project centers on the use of SAT solvers. The concrete task is insprired by the game Reflexion.[1] Given the problem described below, your task is to write a program that

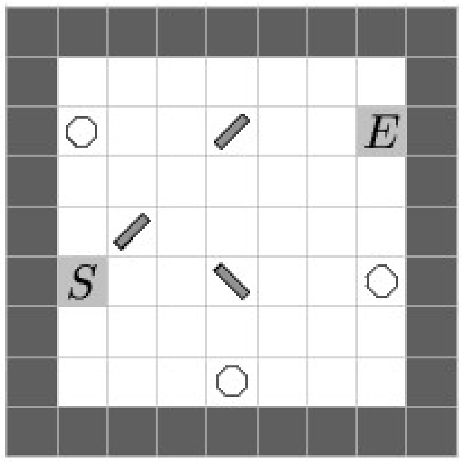
(i) generates a SAT formula that is satisfiable iff the problem instance has a solution;

(ii) use a SAT solver to solve the problem;

(iii) evaluate the output of the SAT solver to either tell the user that there is no solution or to extract the solution from the output of the SAT solver.

1 The problem

Reflexion is a game played on a rectangular grid bordered by walls. There is a designated *start* and *end* position, otherwise each field can be occupied by *mirrors*, *diamonds*, and additional walls. Mirrors are diagonal and thus have two possible orientations.



[1]Originally a PC game by Juho Pohjonen, a browser-playable remake can be found at `http://oos.moxiecode.com/examples/reflex/index.html`.

The game starts by releasing a ball at the start position. The ball travels along the cardinal directions (north, east, south, west); it is reflected from mirrors at an angle of 90 degrees (depending on the orientation of the mirror) and bounces back from walls. The objective of the player is to guide the ball to the end position by manipulating the mirrors, collecting all the diamonds along the way.

For this exercise, we consider a simplified version of the game where

- the start and end positions are inside a boundary wall (hence the initial direction of the ball is implicitly given);
- more importantly, all mirror positions must be fixed before releasing the ball and cannot be changed afterwards.

Given a Reflexion puzzle of this form, the question that the player must solve is therefore:

Is there a possible alignment for the mirrors that will guide the ball from start to end, collecting all the diamonds, and if so, what is it?[2]

2 The input

Your program should take as input a text file describing the puzzle instance. The text file described the puzzle graphically, like in the following example:

```
%%%%%%%%%
%       %
%   M D E
S D M   %
%       %
%%%%%%%%%
```

Here a percentage sign (`%`) means a wall, `S` means the start position, `E` means the end position, `D` means a diamond, and `M` means a mirror. In the example, there evidently exists a solution, by putting both mirrors in "forward" orientation (`/`).

3 The output

The expected output of the program is either the response that there is no solution, or some easily readable representation of the solution, for instance like this:

```
%%%%%%%%%
%       %
%   / D E
S D /   %
%       %
%%%%%%%%%
```

[2]This problem is in fact known to be NP-complete, see M. Holzer and S. Schwoon. *Reflections on Reflexion - Computational Complexity Considerations on a Puzzle Game*, Conference on Fun with Algorithms, 2004, pages 90–105.

4 Your program

Your program can be written in any language you like. You may also write multiple programs, tied together, e.g., with a shell script. Your program ought to generate input for the Z3 solver, either by feeding it input in DIMACS format[3] or by using one of the frontends provided in the previous session. Using DIMACS is preferred, use the frontends at your own risk (they are preliminary and not guaranteed to work well for large instances).

5 Evaluation

Organisation

You may work in groups of two.

Report

A report (in PDF) not exceeding 2 or 3 pages is expected to explain your design choices, how to compile, run, and interpret your solution, and eventual flaws of your program.

Program

Your program will be mostly evaluated with respect to how well it satisfies the required functionality. Other criteria are the organisation of the code and user interface (which may be textual, but is it easy to use, does it present the solution properly etc).

Soutenance

The *soutenance* takes around 10 minutes per solution and will consist of a demonstration of your program and eventual questions. The soutenance will be grouped together with that for the next part (SMT).

Important Dates

Deadline for handing in program and report: March 4 until midnight.
Soutenance: During the session of March 6.

[3]To use Z3 with DIMACS data, invoked it with the `-dimacs` option; additionally, if your data comes on standard input, with the `-in` option.