Parity Games

Beau De Clercq

January 21, 2020

Abstract

In this report we will briefly discuss how it is possible to implement an easy way to solve parity games. Firstly we will describe the models that were used in this implementation, followed by a brief description of how the algorithm itself solves our game. We conclude with some examples input and output.

1 The application

1.1 Models

In this implementation two models were used: a **ParityNode** class and a **ParityGame** class.
As the name suggests, a **ParityNode** object is used to store all information pertaining to the node as described in the input file. Such an object thus has an ID, priority, owner, a list of all successors and optionally a name.
An instance of the **ParityGame** class is used to store all nodes and edges who together form a single game. It consists of a list containing all nodes V, two lists V_0 and V_1 consisting of all nodes belonging to $player_0$ or $player_1$ respectively, a list of edges E and a priority mapping Ω.

1.2 Algorithm

In our application we decided to implement the algorithm by Zielonka as described in [1, p28]

2 Appendix

2.1 PRISM code (ex 6.2)

```
dtmc

module ex2
// local state
s : [0..4];
a : bool;
```

```
b : bool;

[] s=0 -> (s'=1) & (a'=true);
[] s=0 -> (s'=4) & (b'=true);
[] s=1 -> (s'=2) & (a'=true);
[] s=1 -> (s'=2) & (b'=true);
[] s=2 -> (s'=3) & (b'=true) & (a'=false);
[] s=3 -> (s'=3) & (b'=true);
[] s=3 -> (s'=0) & (b'=false);
[] s=4 -> (s'=4) & (b'=true);

endmodule

init
(s=0 | s=3) & a=false & b=false
endinit
```

References

[1] Guillerma A. Pérez, *PLAYING GAMES TO SYNTHESIZE REACTIVE SYSTEMS*