Algorithms for Model Checking
Practical Assignment II

Dr. T.A.C. Willemse, MF 6.073

- you are expected to work in groups of two-three students.
- clearly indicate the contributors to the assignment, *i.e.*, provide your name, student number and email address.
- each group member is considered responsible for the report and results of the group.
- Hand in your report **which includes a link to source code** via Canvas; executables (if present), also via link to download. executables (if present) via link to download. **Only reports in PDF format are accepted.**

General

The assignment consists of five questions. The deadline for handing in the report is Friday 13 January (23.59 o'clock). Any additional models and/or formulae used in your experiments must be **described formally**, and you should **explain** their structure and **argue** their correctness by relating their formal definitions to the informal problem statements (brevity in explanation and arguments is appreciated but not at the expense of clarity). The relevant output produced by the programs must be contained in the report.

Your solution will be graded on a scale of 1-10; see the rubric attached at the end of this document for indications for the grade build-up. In particular, the following are appreciated:

- High report quality:
  - well-structured report (introduction, contents, conclusions, references)
  - clear and concise descriptions of the formalisation of the problems, the explanations of their solutions, your heuristics and their implementation.
- Correctness of results:
  - **in the report**, explain how the pseudo code is implemented (include the **pseudo code**); that is, motivate, **in your report**, your choice of data structures and operations on these; concisely explain how snippets of your code implement the relevant and essential parts of the pseudo code (however, do not explain boilerplate code).
  - practical performance of your code and scalability are *not* the primary concerns; needless computational overhead will, however, be taken into account.

In addition, up to **1 bonus points** may be obtained when you present one or more non-trivial model checking problems of your own choice and show how you can solve it with your solver (see the note in Appendix A), or when you also implement an alternative parity game solver. The final

grade for the assignment is obtained by the grade you receive for the assignment (as given by the rubric), plus the bonus (if applicable), of course with a max of 10 points in total for the assignment.

Assignment

Observe that in the small progress measures algorithm the choice of the next vertex that is considered for lifting is non-deterministic. For this assignment you must consider a number of different lifting strategies and implement these.

1. Implement the small progress measures algorithm for solving parity games. Ensure that your implementation:

   (a) can read parity games in the PGSolver format;[1]

   (b) treats the parity games as **min** parity games, *i.e.* implements the small progress measures algorithm as described in the lecture (**note:** all tools in the PGSolver toolkit generate and use **max**-parity games, whereas you are required to use and solve **min**-parity games).

   (c) implements at least the following two lifting strategies:

   Input order lift the vertices in the parity game in the order they appear in the input (vertices are represented by numbers in the PGSolver format); *i.e.* if the input contains vertices $0, \ldots, 10$ (in that order), start with lifting 0, then 1, up to 10, then restart at 0 again, until the measures stabilise.

   Random order lift the vertices in the parity game in randomised order, *i.e.* fix a random order before starting your algorithm, and then iteratively lift vertices in this order, until the measures stabilise.

   Include the code of the algorithms in an appendix (file reading and other auxiliaries may be omitted from the appendix), and **include all relevant source files** when you send in your report. Make sure your code compiles/can be executed on Linux or Mac OS X (*e.g.* include .jar files for Java and which Java version is required).

2. Some structural properties of parity games could be used to improve the lifting strategies of the small progress measures algorithm, *e.g.* some self-loops and cycles quickly proceed to $\top$, when lifted repeatedly.

   (a) Formulate and implement at least two additional lifting strategies, and clearly explain, **in your report**, why (and when) these strategies make sense (*e.g.*, under which circumstances do these strategies minimise the number of liftings required), and, if relevant, discuss the run-time complexity of computations underlying your lifting strategies.

   (b) Concisely describe how you implemented these lifting strategies.

   Include the code of the heuristics in an appendix.

3. Define 8 test cases (of which at least 6 non-trivial ones; *i.e.* with > 2 different priorities and both players owning at least one vertex) of parity games of up-to 8 vertices and state for each parity game the **winning partition** according to the theory, as well as given by your implementation. Ensure that your test cases cover all operations of the SPM algorithm and some test cases illustrate the role of the special value $\top$.

[1] The PGSolver format is described in the documentation at:
`http://www.win.tue.nl/~timw/downloads/amc2022/pgsolver.pdf`, Section 3.5

4. Download the following three problem sets, and for each of the following parity games, state the winner of the vertex with index 0 and, for both players, how many vertices they win.

   (a) Dining philosophers.[2] These are the parity games that correspond to the model checking questions from Assignment I. For each parity game, state the winner of the vertex with index 0 and how many vertices are won by player even and player odd.

   (b) German's cache coherence protocol.[3] These are the parity games that correspond to two of the model checking questions from Assignment I. For each parity game, state the winner of the vertex with index 0 and how many vertices are won by player even and player odd.

   (c) Elevator.[4] These are parity games that can be generated using the `elevatorverification` tool in the PGSolver toolset. For each parity game, state the winner of the vertex with index 0 and how many vertices are won by player even and player odd.

5. Visualise and analyse (discuss) the performance of *all lifting strategies you implemented* on *all parity games* that you solve. In case your implementation of the 'random' strategy sets a different random seed each time the algorithm is run, you may (should) wish to present averages and outliers as well.

[2] `http://www.win.tue.nl/~timw/downloads/amc2022/dining_games.tar.bz2`
[3] `http://www.win.tue.nl/~timw/downloads/amc2022/ccp_games.tar.bz2`
[4] `http://www.win.tue.nl/~timw/downloads/amc2022/elevator_games.tar.bz2`

A Note on Generating Parity Games

If you want to generate your own parity games, it is important to consider the following:

- The PGSolver tools can be downloaded from the following URL:

  `https://github.com/oliverfriedmann/pgsolver`

- Max-parity games can be converted into min-parity games using the command `transformer -mm <max-pg> > <min-pg>`.[5]

- You can use the tools `lts2pbes` and `pbes2bes` in the mCRL2 toolset[6] to generate *max*-parity games for model checking μ-calculus properties on labelled transition systems. The parity games for the dining philosophers problem set in the assignment have been created using the following commands:

  ```
  lts2pbes -D data.mcrl2 -f <property>.mcf <spec>.aut <spec>.<property>.pbes
  pbes2bes -opgsolver <spec>.<property>.pbes <spec>.<property>.maxpg
  transformer -mm <spec>.<property>.maxpg > <spec>.<property>.gm
  ```

  Note that this requires that the file `data.mcrl2` contains a specification of all actions that are used in the labelled transition system. In this case, `data.mcrl2` contains the following:

  ```
  act plato, others, i;
  ```

- If you use any of the tools mentioned above, include the exact version number of the tool that you have used in your report.

[5] transformer is a tool in the PGSolver toolkit

[6] mCRL2 can be downloaded from `http://www.mcrl2.org`

GRADING RUBRIC

Note: if your program correctly passes all your test cases (see question 3), the grade is determined by the rubric; otherwise the grade is at most a 5.

% of grade	excellent (10)	good (8)	fair (6)	needs improvement (4)	fail(0)
10%: report general appearance	The report serves as a model of how to fulfil the assignment	Writing may contain minor errors but ideas are communicated clearly and the text has been carefully proofread. Visual aids such as figures and tables are used well.	Writing contains frequent errors in spelling, grammar or sentence structure, distracting readers. Weak use of visual aids such as figures and tables.	Writing contains numerous errors in spelling, grammar, or sentence structure which interfere with comprehension. No proper use of visual aids such as figures and tables.	There is essentially no report.
25%: parity game solver design and quality	The report contains an exemplary description of the data structures and techniques used to implement the essential parts of the core parts of the small progress measures algorithm, and it explains the rationale for choosing these.	The report contains a good description of the essential parts of the implementation of the core parts of the small progress measures algorithm, techniques and data structures used. Some rationale is given.	The report contains some discussion on the implementation or its data structures; little to no rationale is given.	The report contains no description of the implementation of the core parts of the small progress measures algorithm, nor of the data structures used.	There is no implementation.
20%: quality of implementation	The implemented solver and its heuristics correctly solves all the parity games in question 4 of the assignment (for games up-to 50k vertices)	The implemented solver and its heuristics correctly solves most of the parity games in question 4 of the assignment (for games up-to 50k vertices)	The implemented solver and its heuristics correctly solves many of the parity games in question 4 of the assignment (for games up-to 50k vertices)	The implemented solver and its heuristics incorrectly solves most of the parity games in question 4 of the assignment (for games up-to 50k vertices)	There is no implementation.
25%: correctness and explanation of heuristics	The report contains an exceptionally clear explanation of four, or more heuristics, the rationale behind the heuristics, and a discussion on which model checking problems (when viewed as parity game solving problems) would typically benefit from the heuristics.	The report contains a good explanation of four, or more heuristics, a decent explanation of the rationale behind the heuristics, and some discussion on which model checking problems (when viewed as parity game solving problems) would typically benefit from the heuristics.	The report contains an explanation of three, or more heuristics, and may contain an explanation of the rationale behind the heuristics or some discussion on which model checking problems (when viewed as parity game solving problems) would typically benefit from the heuristics.	The report contains no more than two heuristics and there is virtually no discussion on rationale or applicability.	There are no heuristics.
15%: visualisation and explanation of the results	The analysis of the observed performance of the solver and the heuristics uses sound and convincing arguments. The performance of the algorithms is visualised such that they facilitate the understanding and the analysis.	The analysis of the observed performance of the solver and implemented heuristics uses mostly sound and convincing arguments. The performance of the algorithms is often visualised in such a way that they facilitate the understanding and the analysis.	The analysis of the observed performance of the solver and implemented heuristics often fails to put sound arguments forward. The performance of the algorithms is either not visualised in a useful way, or not visualised at all.	There is hardly any analysis of the observed performance of the implemented solver and heuristics. Virtually none of the experiments are accompanied by proper visualisations.	There are no visualisations and analyses.
5%: Discussion and results	The results of the conducted experiments and the general observations are summarised and discussed in detail. There is reflection on the assignment. Design choices and possible alternatives and/or improvements are discussed and proposed.	The results of the conducted experiments and the general observations are summarised and discussed. There is reflection on the assignment.	There is some discussion of results. Some attempt is made to reflect on the assignment.	Results are not discussed. No reflection is made.	There are no results.