Algorithms for Model Checking
Practical Assignment I

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

- you are expected to work in groups of three students.
- clearly mark the contributors to the assignment, *i.e.*, provide your name, student number and email address.
- each group member is considered to be 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. **Only reports in PDF format are accepted.**

General

The assignment consists of two parts. Both parts must be described in a single report. The deadline for handing in the report is Friday 16 December (23.59 o'clock). Any **additional** models and/or formulae (if any) 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 solutions to part I and part II are 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, we expect the following:

- A **concise** but **high quality** report:
  - well-structured report (introduction, contents, conclusions, references) of **max. 15 pages** (excluding appendices, if need be).
  - clear and concise descriptions of the formalisation of the problems and the explanations of their solutions.
- 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 problems of your own choice and show how you can solve it with model checking techniques. The final grade for the assignment is obtained by the grade you receive for part I and II, plus the bonus (if applicable), of course with a max of 10 points in total for the assignment.

Part I

Implement a model checker for the modal μ-calculus in a programming language of your own choice. Both the algorithm that is based immediately on the semantics of the μ-calculus and the improved version by Emerson and Lei must be implemented (see the course's website for the pseudo code of both algorithms). Your implementations *do not* need to work using BDDs; explicitly representing the set of states using some data structure suffices for this assignment. Ensure that your implementation:

- can read labelled transition systems in *Aldebaran* format.[1]
- can read modal μ-calculus formulae given by the following grammar:

  $$f, g ::= \texttt{false} \mid \texttt{true} \mid X \mid \texttt{(}f\texttt{\&\&}g\texttt{)} \mid \texttt{(}f\texttt{||}g\texttt{)} \mid \texttt{<a>}f \mid \texttt{[a]}f \mid \texttt{mu}\ X\texttt{.}f \mid \texttt{nu}\ X\texttt{.}f$$

  here, `a` is an arbitrary lower-case string (*i.e.* `a` $\in$ `[a-z][a-z,0-9,_]`* consists of letters and/or the underscore character) matching an action name and $X \in$ `[A-Z]` is a recursion variable; `&&` is a conjunction and `||` is a binary disjunction (mind the compulsory parentheses!); `<_>` is the diamond modality and `[_]` is the box modality; `mu` is the least fixpoint and `nu` is the greatest fixpoint. **Careful:** in contrast to the assumption in the lectures, the scope of the fixpoint operator here does not extend as far as possible; it only extends to the formula f. That is: here, (`nu` $X.[a]X$ `&&` `mu` $Y.[b]Y$) is a conjunctive formula in which both subformulae are a fixpoint formula; this significantly simplifies parsing. You may assume that all fixed point (sub)formulae bind variables with different names; e.g., (`nu` $X.[a]X$ `&&` `mu` $X.[b]X$) does not occur.
- has a command line interface which permits the user to enter the filenames of a labelled transition system and a modal μ-calculus formula, and which allows for toggling between the naive and the improved model checking algorithm
- passes all provided testcases. Summarise your test results in a clearly marked (sub)section: for each testcase provide **the set of states** satisfying the given formula. [2]

Include the code of the algorithms in an appendix (file reading[3] and other auxiliaries such as parsing should be omitted from the appendix). Include a discussion of your **design decisions**, your choice of language and explain the essential ingredients and your choice of **data structures**.

The code must also be made available via a link (SurfDrive, Google Drive, DropBox, etc.) sent along with the final report. Do not use exotic libraries but make sure your code can be compiled/runs on standard contemporary equipment and software installations (*e.g.* in case of java, include a .jar file) without too much hassle. Make sure your code can be run on either a standard, recent Linux distribution or Mac OS X.

[1] See `https://www.mcrl2.org/web/user_manual/language_reference/lts.html#aldebaran-format` for the grammar and explanation of the Aldebaran format.)

[2] Some testcases include the desired verdict; for others, you need to compute the verdict by hand or otherwise. Test sets can be found here: `https://www.win.tue.nl/~timw/downloads/amc2022/testcases.tar.bz2`

[3] This course is not about parsing. Still, we cannot avoid doing a bit of parsing here. The Aldebaran format is easy enough, but for the μ-calculus you'll wish to resort to a bit of parsing technology to build up a data structure called a *parse tree*. The grammar given here has sufficiently many parentheses for making the parsing 'easy'. Have your model checker work on the parse tree of the formulae. For parsing the given mu-calculus grammar, have a look at the pseudo-code that achieves this: `https://www.win.tue.nl/~timw/downloads/parser.pdf`, courtesy of Maarten Manders.

Part II

Use your model checker to conduct the following experiments. In particular:

- For both algorithms, report on **the total number of fixed point iterations required for each combination of formula and model**.
- **for every formula** you give or are given in these experiments, state its **nesting depth, its alternation depth and its dependent alternation depth**.

1. Download the dining philosophers problem set, containing labelled transition systems modelling the famous dining philosophers problem with $n \in [2, \ldots, 10]$ philosophers and several modal μ-calculus formulae.[4] The LTSs contain actions `plato` and `others`, representing that either Plato or some other philosopher starts eating, and actions `i` representing 'uninteresting' events such as picking up forks.

   Compare and visualise (*e.g.* plot in a graph) the performance of both algorithms you implemented on all formulae and all models included in the problem set (you may, *e.g.* wish to plot the number of iterations for fixpoint computations). State whether the initial state satisfies a given property or not. Explain your observations, and, if possible, use formal arguments.

2. Download the demanding children problem set.[5] The problem set contains labelled transition system models for the demanding children for n children for at least all $n \in [2, ..., 10]$; see lecture 3, Monday 22 November for a Kripke Structure model of the problem for $n = 2$. The LTSs contain actions `ask`, `wisdom`, `playing`, representing state changes for a fixed but arbitrary child, and action `i` for all state changes for all other children; see also the readme included in the problem set.

   (a) Phrase at least four non-trivial modal μ-calculus formulae that can be analysed for all models and explain these formulae. Make sure that not all formulae have the same dependent alternation depth.

   (b) Compare and visualise the performance of both algorithms you implemented on the model checking problems you defined for the demanding children. Again, explain your observations, and, if possible, use formal arguments.

3. In Computer Architecture, *cache coherence* is the uniformity of shared resource data that ends up stored in multiple local caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system (source: Wikipedia).

   Download the cache coherence problem set, containing a model of Steven German's (directory-based) cache coherence protocol[6] for $n \in [2, ..., 5]$ clients and several modal μ-calculus formulae.[7] The LTS contains actions `req_exclusive`, `req_shared`, `exclusive` and `shared`,

[4] See `http://www.win.tue.nl/~timw/downloads/amc2022/dining.tar.bz2`
[5] See `http://www.win.tue.nl/~timw/downloads/amc2022/demanding.tar.bz2`
[6] See `http://www.win.tue.nl/~timw/downloads/amc2022/fmcad2004.pdf`
[7] See `http://www.win.tue.nl/~timw/downloads/amc2022/ccp.tar.bz2`

modelling client 1's request for exclusive or shared access to data, respectively, and the notification that it has such access; action `i` represents all the events we abstract from such as events modelling the granting of exclusive access, invalidating data and other clients' events.

Compare and visualise (*e.g.* plot in a graph) the performance of both algorithms you implemented on all formulae and all models included in the problem set. State whether the initial state satisfies a given property or not. Explain your observations, and, if possible, use formal arguments.

4. Consider a two-player game played on a board of positions $\{(x, y) \mid 0 \leq x \leq N, 0 \leq y \leq N\}$. A round of the game proceeds as follows: a pebble on position (x, y) is moved by the following rule:

   - Player I chooses a vector (u_x, u_y) from $U = \{(1, 3), (2, 1)\}$,
   - Next, player II chooses a vector (v_x, v_y) from $V = \{(2, 0), (1, 2)\}$.

   The new position of the pebble is given by $((x+u_x+v_x) \mod (N+1), (y+u_y+v_y) \mod (N+1))$ and the game continues with a new round from this new position. A *play* starting in a position (x, y) is a maximal (i.e. finite, non-extendable, or infinite) sequence of positions visited by the pebble by playing according to the above rules. A play is won by Player II iff the pebble lands on position (N, N) at some point. Player II wins the game iff she has a strategy so that all plays that start in position $(0, 0)$ are won by player II (*i.e.* regardless of how player I plays). In any other case, player I wins.

   Download the board game problem set.[8] The problem set contains labelled transition system models for the board game for $N = 50, 100, \ldots, 450, 500$. The initial state of each LTS represents the pebble being on position $(0, 0)$. Action `choose1` represents player I selecting a vector; action `choose2` represents player II selecting a vector; action `won` is present exactly in those states in which the pebble is—at the beginning of a round—on position (N, N) and leads to a deadlock state (*i.e.* a state without outgoing transitions).

   (a) Give a modal μ-calculus formula that is true iff there is a *play* starting in $(0, 0)$ that player II can win. Use your model checker to verify which of the board games satisfy this property and compare and visualise the performance of both algorithms on the entire problem set.

   (b) Give the modal μ-calculus formula that is true iff player II wins the game. Explain your formula and state its complexity (*i.e.* nesting depth, alternation depth and dependent alternation depth). Use your model checker to verify which of the board games satisfy this property and compare and visualise the performance of both algorithms on the entire problem set.

   (c) Like question (b) but now we change the rules for winning: player II wins a play iff she *infinitely often* manoeuvres the token to position (N, N). As before, player II wins the game iff she has a strategy for manoeuvring the pebble so that all infinite plays are won by her.

[8] See `http://www.win.tue.nl/~timw/downloads/amc2022/boardgame.tar.bz2`

GRADING RUBRIC

Note: if your program correctly passes all provided test cases (see Part I), 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%: model checker design and quality	The report contains an exemplary description of the data structures and techniques used to implement the essential parts of the core model checking algorithms; it explains the rationale for choosing these and explicitly explains the relation between pseudocode and relevant parts of the implementation.	The report contains a good description of the essential parts of the implementation of the core model checking algorithms, 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 model checking algorithms, nor of the data structures used.	There is no implementation.
25%: quality of implementation	The model checkers correctly answer the model checking problems in part II (for state spaces up-to 100k states)	The model checker correctly answers most of the model checking problems in part II (for state spaces up-to 100k states)	The model checker correctly answers roughly half of the model checking problems in part II (for state spaces up-to 100k states)	The implemented model checking algorithms pass virtually none of the model checking problems of part II (for state spaces up-to 100k states)	There is no implementation.
25%: correctness and explanation of formulae	Formulae provided in problem sets and in response to questions are discussed and the given explanations/interpretations of the formulae are spot-on. Formulae given in response to questions are meaningful and are not all of the same complexity. The number of formulae considered go well-beyond the requested number of formulae.	Formulae provided in problem sets and in response to questions are typically discussed and the given explanations/interpretations of the formulae are largely correct. Formulae given in response to questions are meaningful and are not all of the same complexity.	Formulae provided in problem sets and in response to questions are typically discussed but the given explanations/interpretations of the formulae is often incorrect. Formulae given in response to questions are sometimes not meaningful and only a very large number of these belong to the same complexity classes.	Formulae provided in problem sets and in response to questions are hardly discussed and the given explanations/interpretations of the formulae is mostly incorrect. Formulae given in response to questions are not meaningful or all belong to the same complexity classes.	There is no discussion of any of the formulae and no formulae are given in response to the questions.
10%: visualisation and explanation of the results	The analysis of the observed performance of the implemented model checkers uses sound and convincing arguments. Literature is cited appropriately. The performance of the algorithms is visualised such that they facilitate the understanding and the analysis.	The analysis of the observed performance of the implemented model checkers 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 implemented model checkers 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 model checkers. 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.