Algorithms for Model Checking
Practical Assignment I

Dr. T.A.C. Willemse, MF 6.146

- 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 presentation and results of the group.
- Hand in your (1) **presentation**, the filled out (2) **excel file** and (3) a **link** to source code via Canvas; executables (if present), also via link to download (you may include these links on the first slide in your presentation to make things easier).

General

The assignment consists of two parts, and expects three deliverables: (1) a presentation, (2) an excel file containing your experimental data, and (3) a link to a tool. The deadline for handing in the presentation (in powerpoint or PDF) and the tool is Friday 13 December (23.59 o'clock). In particular, we expect a (roughly) 10-minute presentation highlighting:

- the data structures and operations used in the model checking algorithm,
- the results you obtained, and how these are supported (or not) by the theory,
- explanations of models and formulae that you needed to supply.

Practical performance of your code and scalability are *not* the primary concerns; needless computational overhead **may**, however, be taken into account. This means that your tool should be able to handle models of up-to 100k states.

Assessment

Your solution to the assignment will be assessed in a 20-30 minute meeting. Your grade (a pass/fail grade) is determined by the quality of your presentation, your tool, and the ensuing discussion. A passing grade will translate to 10 points, whereas a fail translates to 0 points. Failing to attend the meeting will result in a fail grade. Grades will be published via Canvas.

The presenter (or the presenters) of the (roughly) 10-minute presentation **will be selected by the assessor** on the spot during the meeting. As a consequence, all group members should be able to give the presentation and be capable of taking over from another presenter when requested to do so by the assessor. During the meeting, the assessor may ask group members to demonstrate the tool and/or run the tool on specific input on the spot on their laptop.

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[2]. 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 the excel file:** for each testcase provide **the set of states** satisfying the given formula[3].

The code must also be made available via a link (OneDrive, Google Drive, DropBox, etc.) sent along with the presentation (e.g., as part of the first slide). 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/tools/lts.html` for the grammar and explanation of the Aldebaran format.)

[2] 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://timw.win.tue.nl/downloads/parser.pdf`, courtesy of Maarten Manders.

[3] 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://timw.win.tue.nl/downloads/amc/testcases.tar.bz2`

Part II

Use your model checker to conduct the following experiments. In particular, for problems 1 and 2 below, **in the excel file:**

- For both algorithms, report on **the total number of fixed point iterations required for each combination of formula and model (up-to reasonable sizes)**.
- For both algorithms, report on **whether the initial state of a model satisfies a formula or not**.
- **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 the performance (*i.e.*, the number of iterations for fixpoint computations) of both algorithms you implemented on all formulae and all models included in the problem set. You should be able to explain your observations, and, if possible, use formal arguments during the oral assessment.

2. 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[5] for $n \in [2, ..., 5]$ clients and several modal μ-calculus formulae.[6] The LTS contains actions `req_exclusive`, `req_shared`, `exclusive` and `shared`, 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 the performance (*i.e.*, the number of iterations for fixpoint computations) of both algorithms you implemented on all formulae and all models included in the problem set. You should be able to explain your observations, and, if possible, use formal arguments during the oral assessment.

3. 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 $U = \{(1, 3), (2, 1)\}$,

[4] See `https://timw.win.tue.nl/downloads/amc/dining.tar.bz2`
[5] See `https://timw.win.tue.nl/downloads/amc/fmcad2004.pdf`
[6] See `https://timw.win.tue.nl/downloads/amc/ccp.tar.bz2`

- 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.[7] 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).

In your presentation, you should:

(a) present a modal μ-calculus formula that is true iff there is a *play* starting in $(0, 0)$ that player II can win. Moreover, you should use your model checker to verify which of the board games satisfy this property and compare and visualise the performance (number of iterations of fixpoint computations) 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.

[7] See `https://timw.win.tue.nl/downloads/amc/boardgame.tar.bz2`