Examination: Modelling of Complex Systems

23/01/2019

This exam consists of two parts:

- **CLOSED BOOK**: a written theoretical part (8pts).
- **OPEN BOOK**: a written exercise part (12pts).

The procedure is as follows:

- **For every question, use a new page. Put your name and student number on each page.**
- Solve the theoretical part. When finished, give your solution to the assistant.
- You will then receive a print-out of your project.
- Then you can take your course materials, and start with the exercise part. Allowed materials are:
  - Copies of slides
  - Course notes
  - Nothing more.

The duration of the exam is at most 4 hours.

Studenten die de nederlandstalige master volgen mogen antwoorden in het nederlands..

Success !

Prof. Marc Denecker

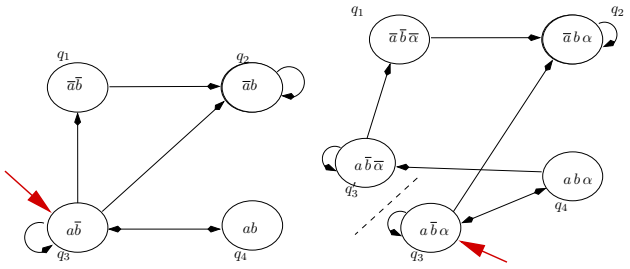
1 Theory part (8pt)

1. Solve either the first question (4pts) or the bonus question (5pts).

   - (4pts) SAT-solving for CNF theories.
     - What is a CNF theory? What is the SAT-problem for CNF theories. Give definitions of unit and conflict clause. Explain Unit Propagation. Formally define input and output of the DPLL algorithm with conflict driven clause learning (CDCL). You do not have to write up the algorithm. Illustrate its operation by executing the following theory until the first conflict is reached after making the following three successive choices: $1, 3, 4$. Explain how backtracking is performed and how search proceeds. (You need only to do the first step - you do not need to compute a model.)

$$
\begin{aligned}
&-1 \vee 2\\
&-1 \vee -4 \vee 5\\
&-3 \vee -4 \vee 6\\
&-3 \vee -5 \vee -6 \vee 7\\
&-7 \vee 8\\
&-2 \vee -7 \vee 9\\
&-8 \vee 10\\
&-1 \vee -9 \vee -10
\end{aligned}
$$

     - Explain alternative strategies of learning clauses, and why the strategy of CDCL is the best.
     - Why does any resolved clause during clause learning contain at least one literal of the last level?
     - Explain some heuristics and optimizations of this algorithm.
   - (5pts) Bonus question. Specify the input and output of the LTL-model checking algorithm in a formal way, give the definitions of $S^\times, \to^\times$ and $C^\times$, and specify the algorithm and its correctness theorem. For an example, see $\mathcal{M}^\times$ for verifying $\phi = \neg a \,\mathrm{U}\, b$ in $s = q_3$ in the context of two transition structures $\mathcal{M}$ and $\mathcal{M}^\times$:



   The desired property of $\mathcal{M}^\times = \langle S^\times, \to^\times \rangle$ is that for each path $(s_1, Q_1) \to (s_2, Q_2) \to \ldots$ of $\mathcal{M}^\times$, $\pi = s_1 \to s_2 \to \ldots$ is a path of $\mathcal{M}$ and for each $\phi \in Q_1$, $\pi \models \phi$.

- Explain how $S^\times$ is constructed.
- Explain how $\rightarrow^\times$ is constructed.
- Show with an example that not every path $(s_1, Q_1) \rightarrow (s_2, Q_2) \rightarrow \ldots$ in $\mathcal{M}^\times$ has the desired property. Indicate how to solve this problem.
- How shall we use $\mathcal{M}^\times$ to check whether $\mathcal{M}, q_3 \models \neg(a \,\mathrm{U}\, b)$

2. Small questions (4pts)

   (a) Given an FO theory T and sentence φ, it holds that T logically entails φ iff $T \cup \{\neg\varphi\}$ is unsatisfiable. Prove this.

   (b) State Gödels incompleteness theorem and explain why it entails undecidability of natural numbers.

   (c) Consider the following LTC:

$$\left\{\begin{array}{l} \ldots \text{frame rules} \\ \forall t(CT\_On(t) \leftarrow \neg On(t)) \\ \forall t(CF\_On(t) \leftarrow On(t)) \\ \forall t(CT\_Off(t) \leftarrow \neg Off(t)) \\ \forall t(CT\_Off(t) \leftarrow Off(t)) \end{array}\right\}$$
$$On(0) \wedge \neg Off(0)$$

   An invariant of this theory is the following:

$$\forall t \ (On(t) \vee Off(t))$$

   Explain why applying the (heavy weight) method for proving invariance on this formula fails. Do you know how to prove its invariance?

   (d) The guard strenghtening condition for an event e that is refined is the proposition expressed below:

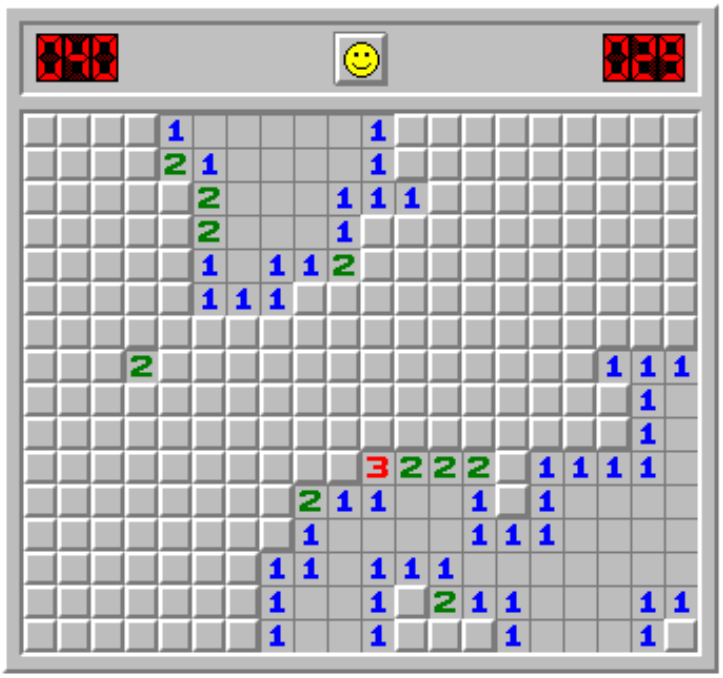
$$\forall x \forall s(A \wedge I(s) \wedge J(s) \wedge H_e(x, s) \Rightarrow G_e(x, s))$$

   Explain the terms in this formula, what it means and what the role of this formula is during the refinement.

2 Exercise part (7+5pts)

Some students have exemption of exercise question 2 or 3 or both. They only make the part of the exam they are not exempted of.

2.1 Minesweeper (5pts)



Most of you know the minesweeper problem. In such problem, the player tries to guess the positions of B bombs on a rectangular $M \times N$ board. He clicks on a cell which then reveals either a bomb in the cell (and then the game is over) or otherwise, the number of bombs in the (at most) 8 surrounding cells. This means that at any time that the game is not lost, the board looks like in the figure. The game is won if every bomb-free cell is clicked, and no bombed cell.

We are modelling one state of the game from the point of view of the player. Given is M, N, B and a predicate that for a number of revealed positions specifies the number of bombs in adjacent positions: *ShownNrOfBombs(row,column,0..8)*.

1. Design a theory T_{mine} about the unknown predicate *BombAt(row,column)*, whose models are exactly all possible configurations of the minefield that are consistent with the information given by the data predicate *ShownNrOfBombs*. You can use FO or any extension that we have seen in the course.

2. Consider the following minesweep-player:

```
Initiate empty partial structure Minefield
While True do {
  compute the certain bomb positions in  Minefield
  compute the certainly safe positions  in  Minefield

  if there is at least one safe position
  then select a safe position; click it
  else select an unknown position;  click it.

  read input
  if Input = ‘‘Won’’
       then Write’’I won!’ ; return
  else if Input = ‘‘Boem!’’
       then  Write ‘‘Ai! I’m dead!’’ ; return
  else add Input  to MineField
       // in the latter case, Input is a set of data items about
       // bomb-free positions and the number of bombs they are
       // surrounded with

}
```

Describe how to implement step 2) of the while loop: the computation of safe positions, using inference on your theory and the current partial structure `Minefield`.

3. Now we change the perspective to the game console. Given a predicate *Click(row,column)* that contains a set of clicked positions and the predicate *BombAt(row,column)*, define the predicate *ShownNrOfBombs(row,column,nr)* which determines which positions are revealed and the number of bombs that surround them. A position is revealed if it is bomb-free and there is a path from a clicked position to it such that all positions on the path except possibly the last one are bomb-free and have zero neighboring bombs.

2.2 Linear Time Calculus: Pushable 1×1 Obstacles (4pts)

In this exercise, you will implement an extension to the first part of the original Rush Hour project and answer some questions about this extension. Those that wish can build from the model solution, at a penalty of loosing 1/3 of the points on that part of the question.

The extension we look at in this exercise, is the addition of a new type of obstacle: pushable 1×1 boulders. A new *push* action is introduced to allow cars to move boulders. A car whose nose or tail touches a boulder can push this boulder over a distance of 1 tile. After the push, both the pushing car and the boulder have moved over 1 tile in the direction of driving. A boulder cannot be pushed if this would cause the boulder to leave the board or to overlap with another board entity. A car can push only one obstable; two or more adjacent

boulders are too heavy to be moved in the direction of adjacency, i.e., the second boulder can be considered to be an entity preventing the first boulder from being pushed.

1. Model the extension in the LTC formalism.

   In order to model this extension, you use the following additional vocabulary (a line starting in an asterisk is a line that was present in the original assignment but has been altered):

   ```
   LTCvocabulary  PushVoc {
       type Boulder isa Entity
       CanPush(Time,Car,Boulder)

       *type Action constructed from
           { M(Car, Dir, Distance),P(Car,Boulder) }
       *ActionAt(Time,Action)
   }
   ```

   with the following intended interpretation:

   Boulder is the subset of all entities that are pushable obstacles. As is the case with other subtypes of `Entity`, a boulder's initial position is given by `Init_Pos(e)` and tracked throughout time by `GetPos(t,e)`. As was the case before, boulders should initially not overlap with other entities.

   CanPush(t,c,b) holds iff. car c can push boulder b over 1 tile at time t. Pushing is only possible if all constraints mentioned above are satisfied.

   (adapted) Action is the renamed version of the type Move from the original assignment. Represents either a move `M(Car,Dir,Distance)` as before, or a push `P(Car,Boulder)` by a car against a boulder.

   (adapted) Actionat(t,a) is the renamed version of `MoveAt(t,m)` from the original assignment. Holds iff. a is the action performed at time t (if there is any action).

   You are also allowed to define any auxiliary predicates and functions (but **not** types) you might require to complete this assignment. Do not forget to change **any** pre-existing aspects of your theory that require changing due to this extension and discuss everything you change.

2. Inferences: Concisely formulate an answer to the following questions:

   (a) Consider the situation where we have a winnable structure containing no boulders. Outline a procedure that finds some location where a new boulder can be placed without making the input structure unwinnable, places a boulder there, and iterates this process until such a location for a new boulder cannot be found. What kind(s) of inference do you use? What extensions or updates are needed to your FO(·)-theory?

(b) Formalize the statement that once a boulder is on the edge of the board, it will never leave this edge in FO($\cdot$). How would you verify this invariant for a given structure in IDP? You are allowed to define an auxiliary predicate, if necessary.

2.3 Temporal logic and Refinement (3pts)

1. Express the following sentences (not necessarily valid) both in LTL and CTL (if it is possible). For a car c, use the predicates `driveLeft(`c`)`, `driveRight(`c`)`, `driveUp(`c`)` and `driveDown(`c`)` to mean that in the current state we will apply the events with the same name. Use the predicate `drive(`c`)` to denote the disjunction of the four drive predicates. Use the predicate `adjacent(`c`,`d`)` to denote that the cars c and d are adjacent. Use the functions `carPos(`c`)` and `goalPos(`c`)` to denote the position of the car c, respectively, the goal position of the car c. If you experience ambiguity, explain what you understood by the sentence.

   (a) If a car c reaches its goal, it can stay there forever

   (b) It is possible that there always will be a car adjacent to c in the future.

   Assume we want to express that every car reaches its goal position and stays at this place forever.

   Is it correctly expressed by the following formulas, and why?

$$\forall c[car] : \mathrm{AF}\,\mathrm{AG}\, carPos(c) = goalPos(c)$$

$$\forall c[car] : \mathrm{F}\,\mathrm{G}\, carPos(c) = goalPos(c)$$

2. Refinement: how does the following proposed updated specification relate to the original assignment: do both refinement properties hold?

   - Each drive event can now take an additional parameter that indicates the length the car drives (in the original assignment this is always 1)