EXAMINATION

Modeling and Analysis of Concurrent Systems 2

course code: 192135320
date: February 3, 2012
time: 13.45–17.15

EXAMPLE ANSWERS

General

- This is an 'open book' exam. Printed handouts (slides, articles, book chapters) may be used, but no handwritten notes, previous examinations, or their answers.
- This exam consists of 3 pages with 5 questions. In total 100 points can be earned: 70 points with the material of the first four lectures (Question 1-4) and 30 points with the material from the research papers (Question 5).

Question 1 (12 points)

Consider the following formulas from CTL*:

1) $\mathbf{AG}\,\mathbf{AX}\,p$ 2) $\mathbf{A}\,\mathbf{GX}\,p$ 3) $\mathbf{AG}\,\mathbf{EX}\,p$

A. *(4 pts)* Indicate which of the three formulas above are in LTL and which are in CTL.

B. *(8 pts)* For each pair of formulas (1,2) and (2,3), check if they are equivalent or not. If so, explain why, referring to the semantics of CTL* formulas. If not, draw a concrete Kripke structure that distinguishes them.

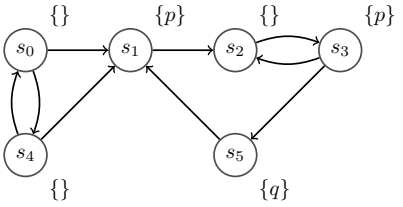
Answer to Question 1

A. CTL: 1) and 3). CTL: 2).

B. 1) and 2) are equivalent: Both force that p holds in all states except the initial state.

Question 2 (20 points)

Consider the CTL formula $\phi := \mathbf{AF}\,(p \wedge \mathbf{EF}\,q)$ and the following Kripke structure K:



A. *(4 pts)* Transform ϕ into an equivalent formula, using only the temporal operators $\mathbf{EX}$, $\mathbf{EG}$ and $\mathbf{EU}$.

B. *(10 pts)* Demonstrate the symbolic model checking algorithm to determine in which states of K does ϕ hold.
You must show which fixpoints are computed, and the results of intermediate iterations.
Transformation to BDDs is not needed; you may use set-notation or logical formulas.

C. *(6 pts)* We now add the fairness constraint $p \vee q$. Indicate the set of states in K where ϕ holds, under the above fairness constraint.
Please give a short explanation, but rerunning the symbolic algorithm is not requested.

Answer to Question 2

A. $\neg\mathbf{EG}\,\neg(p \wedge \mathbf{E}[true\,\mathbf{U}\,q])$

B. Compute inside out, first $\mathbf{E}[true\,\mathbf{U}\;q]$:
$lfp(Z \mapsto q \vee (true \wedge \mathbf{EX}Z)) = lfp(Z \mapsto q \vee \mathbf{EX}Z)$

$$\begin{aligned} false &= \emptyset \\ q \vee \mathbf{EX}\emptyset &= \{s_5\} \\ q \vee \mathbf{EX}\{s_5\} &= \{s_3, s_5\} \\ q \vee \mathbf{EX}\{s_3, s_5\} &= \{s_2, s_3, s_5\} \\ q \vee \mathbf{EX}\{s_2, s_3, s_5\} &= \{s_1, s_2, s_3, s_5\} \\ q \vee \mathbf{EX}\{s_1, s_2, s_3, s_5\} &= \{s_0, s_4, s_1, s_2, s_3, s_5\} \\ q \vee \mathbf{EX}\{s_0, s_4, s_1, s_2, s_3, s_5\} &= \{s_0, s_4, s_1, s_2, s_3, s_5\} \end{aligned}$$

Next, compute $p \wedge \mathbf{E}[true\,\mathbf{U}\,q]$: $\{s_1, s_3\} \cap \{s_0, s_4, s_1, s_2, s_3, s_5\} = \{s_1, s_3\}$. Its negation is $S = \{s_0, s_2, s_4, s_5\}$. Finally, compute $gfp(Z \mapsto S \wedge \mathbf{EX}Z)$:

$$\begin{aligned} true &= \{s_0, s_4, s_1, s_2, s_3, s_5\} \\ S \wedge \mathbf{EX}true &= \{s_0, s_2, s_4, s_5\} \\ S \wedge \mathbf{EX}\{s_0, s_2, s_4, s_5\} &= \{s_0, s_4\} \\ S \wedge \mathbf{EX}\{s_0, s_4\} &= \{s_0, s_4\} \end{aligned}$$

So ϕ holds in the complement of $\{s_0, s_4\} = \{s_1, s_2, s_3, s_5\}$.

C. All states satisfy ϕ under fairness. Every fair path goes through $\{s_1, s_3, s_5\}$ infinitely often, so the s_0, s_4-loop is excluded. Hence every fair path hits s_3, where $(p \wedge \mathbf{EF}\,q)$ holds (even under fairness).

Question 3 (20 points)

A. *(6 pts)* Provide OBDD representations of the formulas $(p \leftrightarrow q) \leftrightarrow r$ and $p \rightarrow (p \wedge q)$.
Use $p < q < r$ as variable ordering.

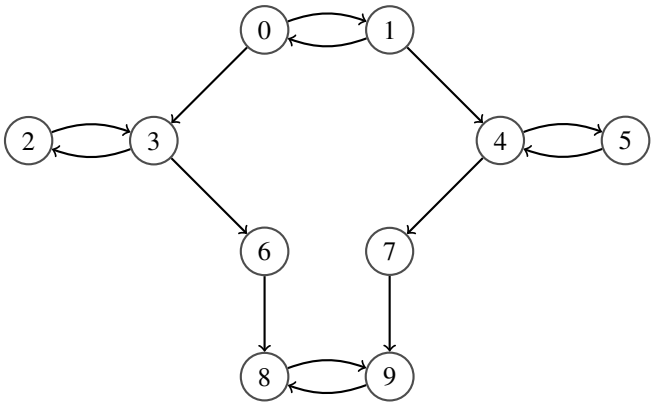
B. *(6 pts)* Demonstrate the APPLY function when computing the conjunction ($\wedge$) of the formulas above. It is sufficient to draw the call graph and the final result.

C. *(4 pts)* Why is using the Operation Cache (also known as Table of Results) essential in the APPLY algorithm in general? Will it help in the previous computation?

D. *(4 pts)* Mention two advantages of the fact that (given a fixed variable ordering), OBDDs provide a *canonical (unique)* representation of Boolean functions.

Question 4 (18 points)

We consider distributed algorithms to compute the strongly connected components of the following graph:



A. *(10 pts)* The FB algorithm (forward-backward) depends on the choice of a pivot node. Which pivot node(s) should be chosen first, in order to split the graph in as many independent subparts as possible? Explain your answer, and indicate how the graph is split in the first round of FB by your chosen pivot node.

B. *(8 pts)* The OBF algorithm tries to split the graph in many layers in one go. Indicate the **O**- and **B**-identified slices in the graph above that OBF will identify in its main loop, when starting at node 0.

Question 5 (30 points)

Please provide short and to-the-point answers to the following questions:

A. *(5 pts)* How does the symmetry reduction approach of [Ip & Dill, 1996] avoid that the complete state space must be generated prior to symmetry reduction?
Does this approach demand any modeling effort from the user?

B. *(4 pts)* The approach of [Dillinger & Maniolis, 2009] uses Cleary tables and Bloom filters to store a set of elements with a very small memory footprint. Mention two factors that contribute to this small memory footprint. Do they affect completeness of model checking?

C. *(4 pts)* Memoised Garbage Collection (MGC, [Nguyen & Ruys, 2009]) is an incremental garbage collection algorithm inspired by an incremental shortest-path algorithm. Does MGC work *correctly* with circular heap structures?

D. *(4 pts)* In [Pasareanu et al., 2008], the L^* learning algorithm is used in the context of compositional verification. What does the L^* algorithm (try to) learn about a component and its environment?

E. *(5 pts)* Both in [Dillinger & Maniolis, 2009] and [Laarman et al., 2010], hash collisions are resolved by *linear probing*. What is linear probing?
Why is it (supposed to be) efficient, given the memory hierarchy of contemporary hardware?

F. *(4 pts)* [Clarke et al., 2001] use a satisfiability solver as computational engine for bounded model checking. Why is their technique called *bounded* model checking? What is the implication for the end user?

G. *(4 pts)* Software model checking, as in [Beyer et al., 2007] is based on CEDAR, Counter-Example Driven Abstraction Refinement. What are the two different reasons for finding a counter-example in this approach? How are both cases handled further by CEDAR?

Answer to Question 5

A. It only explores newly found states when no symmetric one has been found before (i.e. in the same equivalence class). The user must indicate symmetries by using a special sort for identifiers.

B. Part of the data is not stored, but can be retrieved from the index in the hash table. Also, some data is lost (in the Bloom filters). Only the latter affects completeness of model checking.

C. Yes, it is still correct for circular heap structures (but probably not efficient).

D. It learns an assumption on the environment that is sufficient to prove the component correct.

E. When a hash collision is found it searches the next (or previous) memory location. This is good for the L2 cache, because a consecutive part of main memory is loaded into an L2 cache line upon a cache miss, so linear probing avoids extra accesses to main memory.

F. It only tries to find counter examples of length k (the bound). The user doesn't know when to stop, if no counter example is found.

G. A real counter example: terminate with failure. A spurious counter example because the abstraction was too coarse: continue by refining the abstracted model.