15-414 — Bug Catching — Fall 2006

Instructor: Edmund Clarke
Teaching assistant: Himanshu Jain

Assignment 2

Due date: Thursday, December 7, 2006

1 CTL and LTL

In the following argue whether the given LTL and CTL formula are equivalent or not. If they are not equivalent give a model that illustrates the difference.

(a) **AFG**p and **AF AG**p

(b) **AFG**p and **AF EG**p

(c) **AGF**p and **AG AF**p

(d) **AGF**p and **AG EF**p

(e) **AXF**p and **AX AF**p

2 CTL model checking

Show that $\mathbf{EF}f_1$ is the least fixpoint of the function $\tau(Z) = f_1 \vee \mathbf{EX}Z$. Break your proof into the following parts.

(a) Show that $\tau(Z)$ is monotonic.

(b) Let $\tau^l(false)$ be the limit of sequence $False \subseteq \tau(False) \subseteq \ldots$. Let S denote the set of states and R denote the transition relation of a design. Show that for every $s \in S$ if $s \in \tau^l(false)$, then either $s \models f_1$ or there is a state s' such that $(s, s') \in R$ and $s' \in \tau^l(false)$.

(c) Use the above lemmas to show that $\mathbf{EF}f_1$ is the least fixpoint of the function $\tau(Z) = f_1 \vee \mathbf{EX}Z$.

You may want to look at Section 3.7 of Huth and Ryan or Chapter 6 of the Model Checking book to review the terminology and related proofs.

3 Traffic light controller

Recall the traffic light controller discussed in class. We had a `go` variable for north, south, and west direction. The `go` for a particular direction is set to one to indicate a green light in that direction. The SMV code for various versions of traffic light controller is available on the course webpage.

In this problem we want to extend the traffic controller so that it controls traffic in all four directions. Your solution should use processes and have the same fairness constraints as the solution discussed in class.

Specify the following informal specifications using CTL/LTL and check your specifications using a model checker.

(a) Mutual exclusion: there is no traffic on the north/south and east/west lanes simultaneously.

(b) Liveness: If there is traffic in a particular lane, then the `go` variable will eventually become true in that direction.

(c) Non-blocking: It should be possible to make a request to get a green light in a particular direction (a request to go into critical section can always be made).

(d) Strict sequencing: the processes go into the critical section in a particular sequence. We do not want a solution with strict sequencing.

Email your SMV code with specifications to the TA.

4 Dining Philosophers

Dining Philosophers problem is a classic problem related to multi-process synchronization. There are four philosophers sitting at a round table. Between each adjacent pair of philosophers is a fork. That is, the number of forks and the philosophers in same. Each philosopher does two things: *think* and *eat*. The philosopher thinks for a while, and then stops thinking and becomes hungry. When the philosopher becomes hungry, he cannot eat until he owns the forks to his left and right. When the philosopher is done eating he puts down the forks (critical resource) and begins thinking again (non-critical activity!). The philosophers never speak to each other which creates a dangerous possibility of deadlock in which every philosopher holds a left fork and waits perpetually for a right fork (or vice versa).

Give a solution to the Dining philosophers problem in form of a SMV program using processes. Your solution must satisfy the following specifications.

(a) Mutual exclusion: Two philosophers do not own the same fork at any given time.

(b) Liveness: If a philosopher is hungry, then he eventually gets access to the fork to his left and right. In other words, there is no starvation.

(c) Non-blocking: At any time a philosopher can make a request to eat.

(d) Strict sequencing: the processes go into the critical section (acquire forks) in a particular sequence. We do not want a solution with strict sequencing. This is because some of the philosophers might be fast thinkers and fast eaters, while the others may be slow. By having a strict sequencing we might unnecessarily delay the fast philosophers.

Email your SMV code with specifications to the TA.