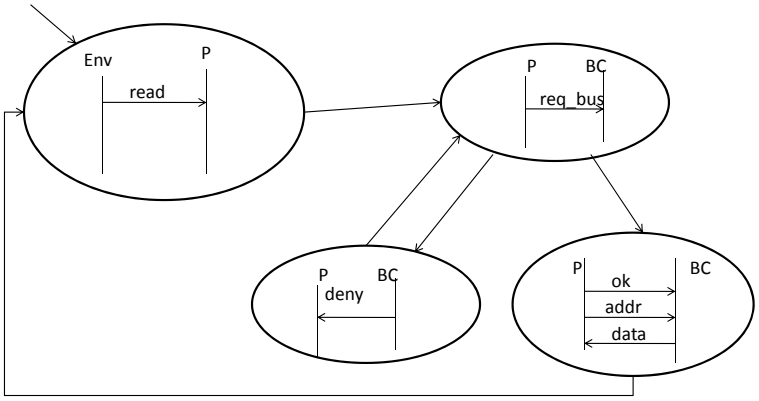
NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 2, 2009/2010
Solutions to CS 4271 - CRITICAL SYSTEMS AND THEIR VERIFICATION

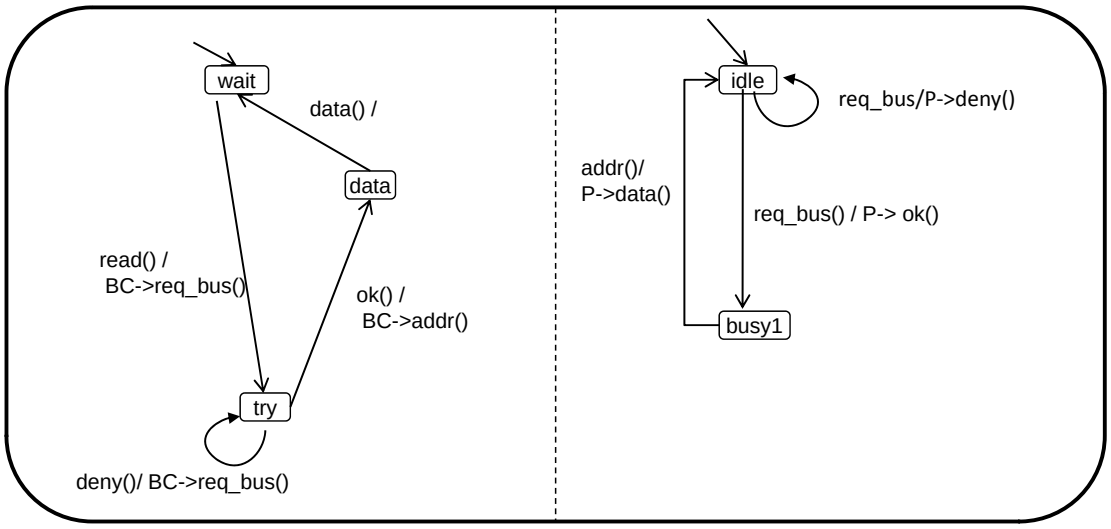
May 2010 Time Allowed: 2 Hours

Question A (Modeling). 8 marks

Consider the following interactions between a processor P and a bus controller BC, modeled by a graph where each node of the graph is a UML Sequence Diagram. You may assume synchronous concatenation of nodes for understanding the graph. Draw the behavior of the "system" as a UML State Diagram. Both the processor and the Bus Controller should appear inside the system, whereas the read requests come from the environment. All communication should be via method calls.

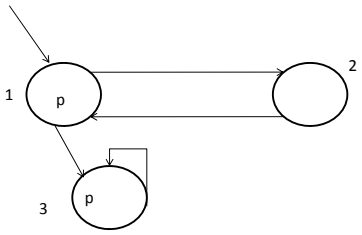


Answer:



Question B (Model Validation). 4 + 6 = 10 marks

Question B.1 Does the following state machine satisfy FGp? Does it satisfy GFp? Give justification of your answer.



The states are numbered 1,2,3. States 1 and 3 satisfy atomic proposition p while state 2 does not. State 1 is the initial state as shown.

Answer:

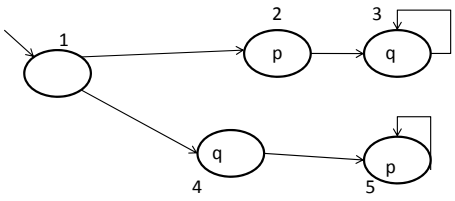
The traces of the state machine are $(1,2)^\omega$ and $(1,2)^*1,3^\omega$. Both of these traces satisfy GFp while $(1,2)^\omega$ does not satisfy FGp. So, the state machine satisfies GFp, but it does not satisfy FGp.

Question B.2 Are the following Linear Time Temporal Logic formulae equivalent? If yes, give a formal proof of equivalence. If not, give an example state machine which satisfies one property but does not satisfy the other. Assume p, q are atomic propositions.

- $F(p \wedge q)$ and $Fp \wedge Fq$
- $G(p \wedge q)$ and $Gp \wedge Gq$

Answer:

- The formulae $F(p \wedge q)$ and $Fp \wedge Fq$ are not equivalent. Consider the following state machine.



  The traces of the state machine are 123^{ω} and 145^{ω}. Clearly the trace 123ω satisfies Fp since the state 2 satisfies p. Similarly the trace 123ω satisfies Fq since the state 3 satisfies q.

  Clearly the trace 145ω satisfies Fq since the state 4 satisfies q. Similarly the trace 145ω satisfies Fp since the state 5 satisfies p.

  Thus, both the traces of the state machine satisfy $Fp \wedge Fq$. However, none of the traces of the state machine satisfy $F(p \wedge q)$ — there is no state in the state machine which satisfies $p \wedge q$.

- The two formulae $G(p \wedge q)$ and $Gp \wedge Gq$ are equivalent. Consider a trace π satisfying $G(p \wedge q)$. Clearly all states in the trace satisfy $p \wedge q$. This means that all states on π satisfy p as well as q. So, $\pi \models Gp$ and $\pi \models Gq$. Thus, $\pi models Gp \wedge Gq$.

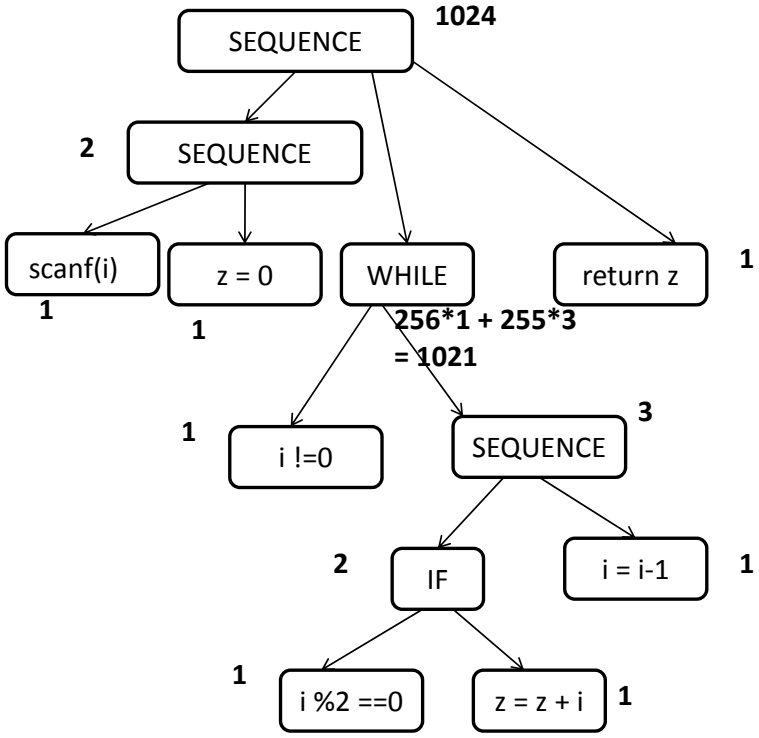
  The proof in the other direction is similar. Again we consider a trace π such that $\pi \models Gp \wedge Gq$. This means that all states on the trace satisfy p as well as q, that is, all states on the trace π satisfy $p \wedge q$, thus the trace $\pi \models G(p \wedge q)$.

Question C (Software Performance Validation). 6 + 8 = 14 marks

Question C.1 Consider the following program fragment, where the input i is an unsigned integer which can be represented via one byte. Using Timing Schema WCET analysis method discussed in class, derive the maximum execution time of the program fragment. Show the abstract syntax tree as well. Assume that each assignment/return/condition-evaluation/scanf takes 1 time unit.

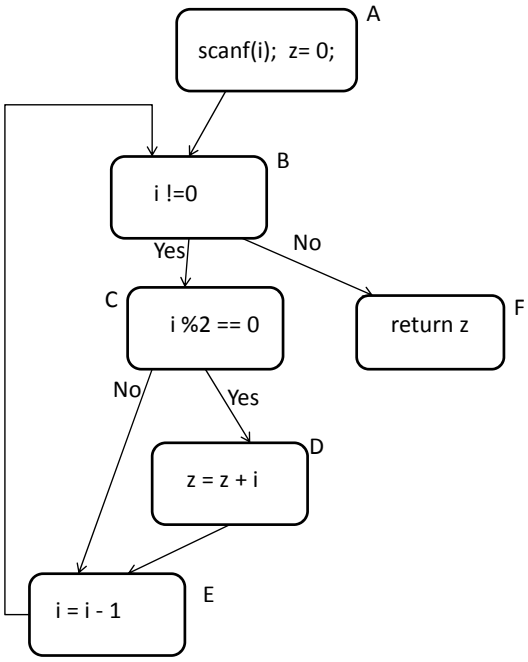
```
scanf("%u", &i); z = 0;
while (i !=0){
        if (i %2 != 0){ z = z + i; }
        i = i - 1;
}
return z;
```

Answer: Since the input i is represented as a one byte unsigned integer it can take the values 0..255. The loop bound is 255. The abstract syntax tree, as well as the timing calculation (via a bottom-up pass of the abstract syntax tree) are shown in the following. The time estimate is 1024 time units.



Question C.2 Formulate the maximum execution time estimation of the program fragment in Question C.1 using Integer Linear Programming (ILP). Construct the control flow graph, show the objective function and show all constraints. Your ILP problem should only perform program path analysis and not micro-architectural modeling. You should try to encode constraints capturing infeasible path information which can help you obtain tighter time estimates.

Answer: The control flow graph is shown in the following.



The basic blocks are numbered A,B,C,D,E,F. The objective function being maximized is $2N_A + N_B + N_C + N_D + N_E + N_F$. N_i denotes the execution count of basic block i. $X_{i,j}$ denotes the execution count of the edge from basic block i to basic block j.

The flow constraints are as follows.
$1 = N_A = X_{A,B}$
$N_B = X_{A,B} + X_{E,B} = X_{B,C} + X_{B,F}$
$N_C = X_{B,C} = XC, E + X_{C,D}$
$N_D = X_{C,D} = X_{D,E}$
$N_E = X_{C,E} + X_{D,E} = X_{E,B}$
$N_F = X_{B,F} = 1$
The loop bound is given by the constraint $X_{E,B} \leq 255$.

We also note that basic block D cannot be executed in two consecutive iterations. Thus B,C,D,E,B,C,D is an infeasible path. Similarly, BCEBCE is an infeasible path. Such infeasible paths are hard to detect. However we can encode them as the following constraints

$X_{C,D} \leq 128$ and $X_{C,E} \leq 128$.

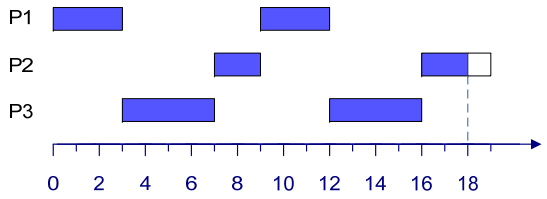
Question D (System Performance Validation). 4 + 4 = 8 marks

Question D.1 Consider three periodic tasks P1 = (3,9,9), P2 = (6,18,18), P3 = (4,12,12). For each task, we have given the (execution time, period, deadline) where period = deadline. All tasks arrive at time 0.

Try to construct a RMS schedule for this task set. Can RMS schedule these tasks?

Answer:

Following is the RMS schedule. Static priorities are assigned $P1 > P3 > P2$. Process P2 cannot meet its deadline, at time == 18, only four of its six units of computation has been executed.



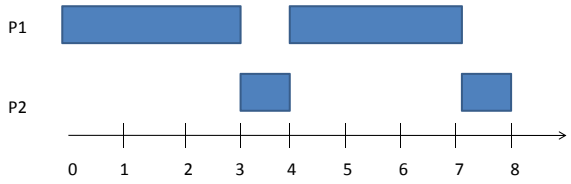
Thus, RMS cannot schedule this task set.

Question D.2 The processor utilization is defined as $\sum_{1 \leq i \leq n} c_i/p_i$ where n is the number of tasks in the task-set, c_i is the WCET of task i, and p_i is period of task i. The Rate monotonic scheduling (RMS) method is guaranteed to produce a schedule if the processor utilization $U \leq n \times (2^{1/n} - 1)$, that is, this is a sufficient condition for schedulability using RMS. Show that this is not a necessary condition for schedulability using RMS.

Answer:

This is done by constructing a task-set which does not satisfy the above inequality but is schedulable via RMS.

Consider a task-set with two independent tasks P1 = (3,4,4) and P2 = (2,8, 8). Here n =2, and the inequality gives $U \leq 2 \times (2^{1/2} - 1)$ that is $U \leq 2 \times 0.414$ that is $U \leq 0.828$. Here the utilization is $U = 3/4 + 2/8 = 1$ And yet, the task-set is schedulable by RMS as shown in the following.



Question E (Software Functionality Validation). 6 + 4 = 10 marks

Question E.1 Consider the following program fragment. Assume that the input array `a` being processed contains 99 positive integers, followed by a zero. Thus, `sum` returns the sum of the 99 positive integers, and prod (unintentionally) returns zero (whereas it is supposed to return the product of the 99 integers).

```
void main(){
    int a[], i = 0, sum = 0, prod = 1;
    scanf(...) /* read input array a*/
      while (a[i] >= 0){
         sum += a[i]; prod *= a[i]; i++;
       }
     printf("%d,%d\n", sum, prod);
}
```

(i) How would you fix the bug corresponding to this observable error?

(ii) Based on dynamic slices discussed in class, we can define the notion of a dice where $dice(x, y) = slice(x) - slice(y)$ for two slicing criterion x,y. What are the statements in $dice(crit_{prod}, crit_{sum})$ where $crit_{prod}$, $crit_{sum}$ correspond to the slicing criteria for the output variables prod, sum (at the place where they are printed).

(iii) Is the dice useful in this case for locating the actual bug?

Answer:

(i) Change the loop guard to `a[i] > 0`

(ii) The dice consists of only the following statements

```
prod = 0;

prod *= a[i];
```

(iii) Dices are useful when some of the outputs of a program are "unexpected", but many others are as expected by the user. We can then remove the statements in the slices of the "correct/expected" variables from the statements in the slices of the "incorrect/unexpected" variables. The above-mentioned dice is thus useful if the `prod` variable's value at the end of the program is unexpected, while the `sum` variable's value is as expected.

The dice is not useful in this case since the loop guard statement is in the slice of both the criteria.

Question E.2 Consider the following two program fragments. `x` is the integer input and `out` is the integer output of each program fragment.

```
y =2; if (x - y > 0){ out = y;} else { out = 0;}

y =12; if (x - y > 0){ out = y;} else { out = 0;}
```

- Give one sample value of `x` for which both the program fragments produce output 0.
- Characterize the set of all values of input `x` which cause the two program fragments to return different outputs.

Answer:

- `x` $== 1$
- The branch `(x -y) > 0` has to evaluate to false in both program versions for the output to be zero in both programs. If at least one of the branches evaluate to true, the two program outputs are guaranteed to be different. This produces the constraint

  $$x - 2 > 0 \vee x - 12 > 0$$

  that is, $x > 2 \vee x > 12$, that is, $x > 2$.