Embedded Systems

Duration : 3h **All documents allowed**

The three parts are independent. Please answer on 3 separate sheets.
Informal explanations in plain english will be appreciated a lot, and it is compulsory to justify all answers.
The number of points associated with each question is only an indication and might be changed slightly.

Part I - Modeling Time and Concurrency (6 points)

We consider a special kind of **Mealy** automaton (called **ATS** in the sequel), in which states can be *timed*. Figure 1 is an example: the outputs are v, y, z and the inputs are u, x and t; the loop transitions that emit nothing are omitted (e.g., the loop with condition "not u" on A). The timed state is C. Starting from A, when u occurs, the ATS goes to B, emitting v; then, when x occurs, it goes to C, emitting y. From then on, it can stay in C at most until the fourth occurrence of t. If u occurs meanwhile, the ATS goes back to B. But if four occurrences of t have occurred, and no u, since the ATS has entered C, then the special transition denoted by a black semi-circle (the *time-out* transition) is taken, z is emitted, and the ATS goes to A.

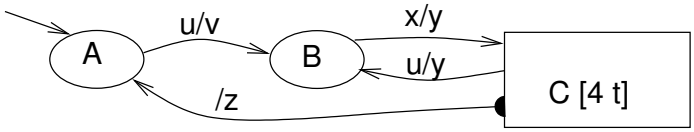


Figure 1: An example Mealy automaton with a timed state C

▷ **Question 1 (1 point) :**

Choose a priority, for the case when the ATS is in C, and u occurs exactly at the same time as the fourth occurrence of t. Draw timing diagrams of the ATS of Figure 1, for a small number of well chosen executions. The idea is to illustrate the time-out behavior, and the priority.

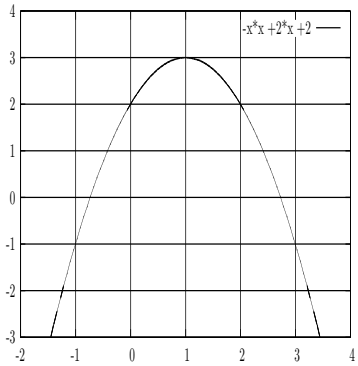
▷ **Question 2 (2 points) :**

— For the example of Figure 1, express the behavior of the ATS as the composition of two ordinary Mealy automata (i.e., without timed states).
— Explain precisely how you compose two ATSes in parallel, synchronously. Notice they do not necessarily have the same input for counting "time".

▷ **Question 3 (3 points) :**

Use several ATSes in parallel to describe the following (hypothetical!) mobile access control system called MACS: *the system is a box with ten buttons* B0 *to* B9 *(it is impossible to press two buttons at the same time); the user walks, carrying the box; he uses the 10 buttons to enter a 4-digit code (say 1234 for this exercice); he should do that "sufficiently fast" and also walking "sufficiently far". If all these conditions are met, the system opens a door (command* open*); otherwise, everything is cancelled, and the user has to start again.*
In order to measure time, there is an external clock ck delivering a signal second to MACS. "*sufficiently fast*" means: (i) the time between two successive buttons should be less than Δ seconds; (ii) the total time between the first button pressed, and the last one, should be less than Γ seconds.
In order to measure distances, there is an external GPS-like system: MACS can start (resp. stop) the GPS with command start (resp. stop). When started, the GPS stores the current position, and then it delivers a signal m to MACS, for each new meter away from this position, until it is stopped. "*sufficiently far*" means that the distance between the position where the first button was pressed, and the position where the last button was pressed, should be more than Π meters.
To summarize, MACS has inputs {B0, ..., B9, ck, m} and outputs {open, start, stop}.

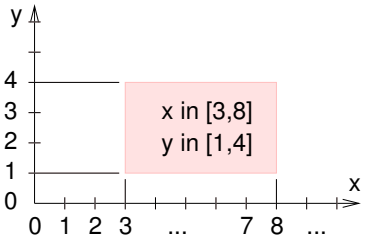
Part II - Abstract Interpretation (5 points)



(a) $f(x) = -x^2 + 2x + 2$

```
program loop2var
 x,y : integer ;
begin
 x := 0 ;
 y := 5 ;
 while (x < y) loop
  assert (2*y -x == 10) ;
  x := x + 2 ;
  y := y + 1 ;
 endloop;
end
```

(b)



(c)

Figure 2: A function for exercice II.1, a program and an abstraction example for exercice II.2

Exercice II.1 - Defining Post Functions

We consider the function : $f(x) = -x^2 + 2x + 2$, whose curve is given on Figure 2-(a).

We are interested in defining precisely the function $\mathsf{post}_{x := f(x)}([a, b])$, for an interval $[a, b]$ of integers where a and b are integer bounds. In other words, we ask the question: if I know that an integer variable x is in the interval $[a, b]$ and I execute the instruction $x := f(x)$, in which new interval $[c, d]$ are the values of x now?

▷ **Question 4 (2 points) :**

— Give a precise definition of $[c, d] = \mathsf{post}_{x := f(x)}([a, b])$, with justifications.

— Do you think your method can be generalized to other functions? Which ones?

Exercice II.2 - Analysing a Loop Program with Intervals

We consider the program of Figure 2-(b). We suggest you run it by hand to get an idea of its behavior. Remember that "`assert C`" is a way of checking the condition `C` at runtime. If the condition happens to be false when entering the loop, the program stops.

▷ **Question 5 (1 point) :**

— Is there a possibility to violate the "`assert`" statement when running the program?

— Build the control graph (or interpreted automaton) of the program, where transitions are labeled by either a condition, or an assignment. Do not forget the `assert` statement.

We aim at running abstract interpretation with intervals on this program. When there are two variables x, y in a program, the interval abstraction of a set of values for the tuple $< x, y >$ is a rectangle, whose sides are parallel to the axes (see example on Figure 2-(c)).

▷ **Question 6 (2 points) :**

— On a diagram with axes x and y (like on Figure 2-(c)), draw the function $y = x$ and the function $y = (x + 10)/2$. Explain why they are meaningful for the program.

— Perform 4 or 5 steps of the analysis with intervals on the program. At each step, you have a rectangle associated with the control state that corresponds to the entry point of the loop. Draw the successive rectangles, using your diagram.

— Explain why the iterative analysis does not stop.

— What about the sequence of intervals associated with the error state due to the `assert` statement? Can you prove that the assertion is never violated?

Part III - Model-Checking (10 points)

1 Explicit versus Symbolic Representation

Consider the Verilog module shown in Figure 3. It is an implementation of a simple vending machine. The machine accepts 20c coins and has two buttons to serve tea or coffee. Tea costs 40c (so two coins are required) and coffee costs 60c (three coins). If no money was inserted into the machine or both buttons are pressed at the same time, then the machine ignores the request.

▷ **Question 7 (2 points) :**

1. Draw the Kripke structure defined by this module. Which states satisfy the formula $\mathsf{EX}(\texttt{state} = \texttt{IDLE} \wedge \texttt{money} = \texttt{FORTY})$?
2. Write an initial and a transition predicate for the module. Give a predicate over the state variables that characterizes all states satisfying Post($\texttt{state} = \texttt{MAKE\_COFFEE}$). (These are all states that can be reached within one step from states satisfying $\texttt{state} = \texttt{MAKE\_COFFEE}$).

```
typedef enum {ZERO,TWENTY,FORTY} amount;
typedef enum {IDLE,MAKE_TEA,MAKE_COFFEE} drink;

module vending(clk, insert_cents, press_coffee, press_tea);
   input clk;
   input insert_cents, press_coffee, press_tea;
   amount reg money;
   drink  reg state;

   initial begin
      money = ZERO;
      state = IDLE;
   end

   always @(posedge clk)  begin
      state = IDLE;
      if (insert_cents) begin
        case (money)
          ZERO: money = TWENTY;
          TWENTY: money = FORTY;
          FORTY : money = FORTY;
        endcase // case (money)
      end
      if (press_tea & ~press_coffee) begin
        case (money)
          TWENTY: begin state = MAKE_TEA; money = ZERO; end
          FORTY : begin state = MAKE_TEA; money = TWENTY; end
        endcase // case (money)
      end
      if (~press_tea & press_coffee) begin
        case (money)
          FORTY : begin state = MAKE_COFFEE; money = ZERO; end
        endcase // case (money)
      end
   end
endmodule // vending
```

Figure 3: An implementation of a simple vending machine.

2 Specifying using Logic

Write logical formulas describing the following properties about our simple vending machine. In the formulas you can refer to the value of a register (as usual) and you can refer to the value of an input variable (unlike in the model checker VIS, which requires to copy the input value into registers).

▷ **Question 8 (2 points) :**

1. Write a CTL formula saying that there is a way (i.e., a sequence of environment actions) to obtain tea from the vending machine (i.e, the machine is eventually in state `MAKE_TEA`).
2. Write an LTL formula stating that tea is never served without inserting money, i.e., if we never insert money (`insert_cents` is always 0), then we are never in state `MAKE_TEA`.
3. Give a CTL formula stating that whenever forty cents were inserted into the machine (i.e., `money` =`FORTY`) and we press the coffee button (`press_coffee` is 1) and we do not press the tea button (`press_tea` is 0), then in all successor states we will make coffee (i.e., `state` = `MAKE_COFFEE`).
4. Give a CTL formula saying that we can reach a state, in which there are twenty cents left in the machine and the machine is serving coffee.
5. Write an LTL formula stating that if we infinitely often insert money into the machine, then we infinitly often make either coffee or tea. Does our vending machine (shown in Figure 3) satisfy this formula? If not, give a sequence of input values showing the violation of the formula.

3 Binary Decision Diagrams

Note that, as usual, we use the shortcut BDD to refer to a Reduced-Ordered Binary Decision Diagram.

▷ **Question 9 (2 points) :**

1. Draw the BDDs representing the Boolean expressions $((b \vee \neg c) \wedge (c \vee d \vee e)) \vee a$ under the variable ordering $a \prec b \prec c \prec d \prec e$ (i.e., a is the top-level variable).
2. List three benefits of using BDDs.
3. Assume we have a Boolean formula f with n variables, $x_1, \ldots, x_n$ and some fixed variable order. How many nodes can a BDD representing f have in the worst case (i.e., give a formula in terms of the number of variables n). Give a family of formulas and a variable order that shows that your previous answer is correct.

4 CTL Model Checking

Consider the Kripke structure K shown in Figure 4 and the fairness constraint $c = \{3, 8\}$ (marked by double cycles). The atomic proposition p is satisfied in state 4, 5, 8 and 9.

▷ **Question 10 (2 points) :**

1. Compute the set of fair states for the fairness constraint c.
2. Are the following statements true or false?
   (a) $K \models \mathsf{EF}(\mathsf{EG}\, p)$
   (b) $K \models_c \mathsf{EF}(\mathsf{EG}\, p)$ under fairness constraint c

   For each formula, justify your claim by computing the set of states satisfying the formula.

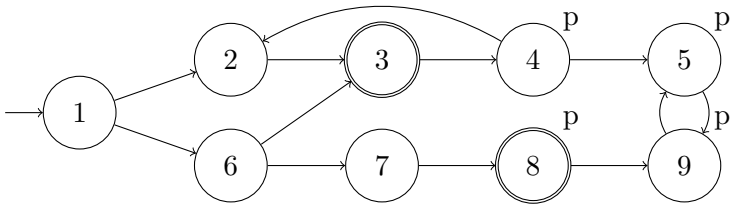


Figure 4: Kripke structure K with fairness condition $c = \{3, 8\}$

5 Büchi Automata and LTL

▷ **Question 11 (2 points) :**

1. Draw a Büchi automaton for $\mathsf{G}(a \rightarrow (a \,\mathsf{U}\, b))$.
2. Does the Kripke structure K' in Figure 5 satisfy the formula $\varphi = \mathsf{G}(a \rightarrow (a \,\mathsf{U}\, b))$? If not, give a behavior (trace) of the structure that violates the formula. Otherwise, describe how you check if $K \models \varphi$ holds using Büchi automata.
3. A *CoBüchi automaton* is a tuple $A = (\Sigma, Q, q_0, \delta, C)$ consisting of a finite alphabet Σ, a finite set of states Q, an initial state q_0, a transition relation δ, and a set of states C, called *CoBüchi states*. A run of A on an infinite word $w_0 w_1 \ldots$ is defined in the same way as for a Büchi automaton. The two automata differ in the way the acceptance condition is defined: a run $\rho = q_0 q_1 \ldots$ of a CoBüchi automaton is accepting if there exists $i \geq 0$ such that $q_j \in C$ for all $j \geq i$, meaning that from some point onwards only states in C are visited.
   Consider the alphabet $\Sigma = \{a, b\}$.
   (a) Draw a CoBüchi automaton A whose language consists of all infinite words with a suffix containing only a symbols, i.e., $L(A) = \Sigma^* a^\omega$.
   (b) Consider the language $(b^* a)^\omega$ consisting of all words with infinitely many a symbols. Can you draw a CoBüchi automaton that accepts this language?
   (c) Is there a language that can be accepted by a Büchi automaton but not by a CoBüchi automaton or vice versa? If yes, give an example for each direction.

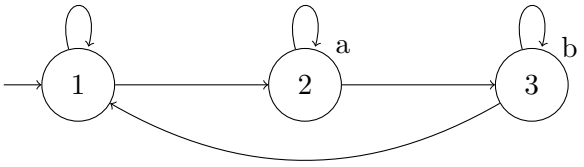


Figure 5: Kripke structure K'