CS738 Assignment 1

Sankalp Gambhir
180260032

November 5, 2020

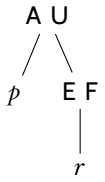
Question 1.

(a) The formulae would be classified as

1. $\mathsf{G}\,\mathsf{F}\,(p \wedge \neg \mathsf{X}\,p)$ — **(A)** It is a well formed LTL over an infinite word, or path in a computation tree, however, due to the use of G, F, X free of state qualifier A wouuld make this (D), not a well formed formula if considered over a transition system (i.e. well formed per the usual LTL grammar, but not well formed as a subset of CTL*)
2. $\mathsf{A} \neg \mathsf{G} \neg p$ — **(C)**
3. $\mathsf{A}\,\mathsf{E}\,\mathsf{F}\,r$ — **(C)** I am unsure about A and E occuring together but the grammar seems to allow it. In its current form it is not a valid CTL formula, however, $\mathsf{A}\,\mathsf{E}$ is functionally equivalent to A so it is equivalent to a CTL formula, same as the previous part.
4. $\mathsf{F}\,(p \Rightarrow \mathsf{G}\,r) \vee (\neg q)\,\mathsf{U}\,p$ — **(A)** Could also be (D), same argument as 1.
5. $\mathsf{A}\,(\,\mathsf{E}\,(p\,\mathsf{U}\,r) \vee \mathsf{X}\,q)$ — **(C)** Use of Next without attached state qualifier.

(b) The formula is $\mathsf{A}\,\mathsf{G}\,(p \Rightarrow \mathsf{A}\,\mathsf{X}\,\mathsf{E}\,[q\,\mathsf{U}\,r])$. Its parse tree is

(c) The formula is $\mathsf{A}\,(p\,\mathsf{U}\,\mathsf{E}\,\mathsf{F}\,r)$. Its parse tree (in CTL) is



Question 2.

The transition system is

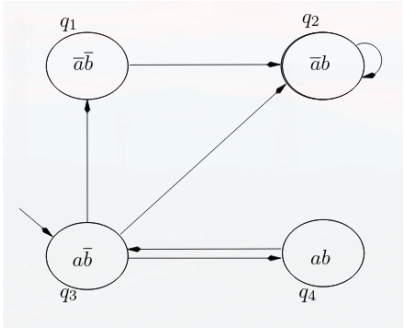


Figure 1: The model

(c) $a\,\mathsf{U}\,\mathsf{X}\,(a \wedge \neg b)$ — A path satisfying this is the trivial $q_3 \leftrightarrow q_4$ loop. q_3 satisfies a and q_4 satsifies $\mathsf{X}\,(a \wedge \neg b)$ by moving to q_3 again. None of the paths with the prefixes $q_3 \rightarrow q_1$ or $q_3 \rightarrow q_2$ satisfy ϕ, hence, $\mathcal{M}, q_3 \not\models \phi$.

(d) $\mathsf{X}\,\neg b \wedge \mathsf{G}\,(\neg a \vee \neg b)$ — A path satisfying this is $q_3 \rightarrow q_1 \rightarrow q_2 \leftrightarrow q_2$. The paths with prefix $q_3 \rightarrow q_4$ do not satisfy the first clause, hence, $\mathcal{M}, q_3 \not\models \phi$.

Question 3.

(a) To prove: $\mathsf{G}\,\phi \equiv \phi \wedge \mathsf{X}\,\mathsf{G}\,\phi$

Consider an infinite path, π represented by the word $\prod_i \geq 0 s_i$ in a model $\mathcal{M}$.

$\Rightarrow$ direction. If $\mathcal{M}, \pi \models \mathsf{G}\,\phi$ then by definition, all infinite suffixes of π satisfy ϕ. In particular, dropping s_0, the path π^1 satisfies $\mathsf{G}\,\phi$, i.e. π satisfies $\mathsf{X}\,\mathsf{G}\,\phi$. Combined with the fact that the trivial zeroth suffix π itself satisfies ϕ, we get the implication.

$\Leftarrow$ direction. If $\mathcal{M}, \pi \models \phi \wedge \mathsf{X}\,\mathsf{G}\,\phi$, then once again by definition of Globally, as well as Next, all infinite suffixes of π^1 satisfy ϕ, which leaves only one suffix of π, which is π^0 or π itself but it satsifies ϕ due to the first clause of our formula. Thus, all infinite suffixes of π satisfy ϕ, which is equivalent to saying it satisfies $\mathsf{G}\,\phi$.

Since this is true for any path π of the arbitrary model, the two given formulae are equivalent.

(b) To prove: $\mathsf{E}\,\mathsf{F}\,(\phi_1 \vee \phi_2) \equiv \mathsf{E}\,\mathsf{F}\,\phi_1 \vee \mathsf{E}\,\mathsf{F}\,\phi_2$

Consider a model $\mathcal{M}$.

$\Rightarrow$ direction. If $\mathcal{M} \models \mathsf{E}\,\mathsf{F}\,(\phi_1 \vee \phi_2)$, then there exists a path π such that $\mathcal{M}, \pi \models \mathsf{F}\,(\phi_1 \vee \phi_2)$. So, there exists a suffix of π which satisfies $\phi_1 \vee \phi_2$, i.e., at the first state of that suffix, one of ϕ_1 and ϕ_2 holds. Suppose ϕ_1 holds. This implies that the path satisfies $\mathsf{F}\,\phi_1$ and so $\mathcal{M} \models \mathsf{E}\,\mathsf{F}\,\phi_1$. By the definition of the Or operator, this Or something trivially holds, and we get our equivalence. A similar argument follows in the case ϕ_1 does not hold but ϕ_2 does on the chosen suffix.

$\Leftarrow$ direction. If $\mathcal{M} \models \mathsf{E}\,\mathsf{F}\,\phi_1 \vee \mathsf{E}\,\mathsf{F}\,\phi_2$, there must exist a path π_1 such that $\mathcal{M}, \pi_1 \models \mathsf{F}\,\phi_1$ or a path π_2 such that $\mathcal{M}, \pi_2 \models \mathsf{F}\,\phi_2$. Suppose the first is true, then there is a state in π_1 such that ϕ_1 holds there. But by the definition of the Or operator, this implies that $\phi_1 \vee \phi_2$ holds at that state, and by extension, $\mathcal{M}, \pi_1 \models \mathsf{E}\,\mathsf{F}\,(\phi_1 \vee \phi_2)$. A similar argument follows again in the case the second (π_2 condition) holds.

Since this is true for an arbitrary model, The two given formulae are equivalent.

Question 4.

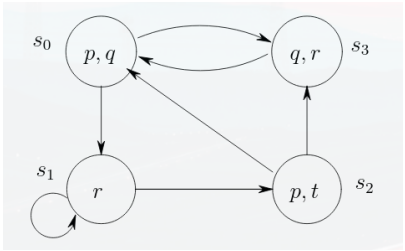


Figure 2: The model

(a) $\mathsf{A}\,\mathsf{F}\,q$ — $\mathcal{M}, s_0 \models \phi,\ \mathcal{M}, s_2 \models \phi$

(b) $\mathsf{A}\,\mathsf{G}\,\mathsf{E}\,\mathsf{F}\,(p \vee r)$ — $\mathcal{M}, s_0 \models \phi,\ \mathcal{M}, s_2 \models \phi$

(c) $\mathsf{E}\,\mathsf{X}\,\mathsf{E}\,\mathsf{X} r$ — $\mathcal{M}, s_0 \models \phi,\ \mathcal{M}, s_2 \models \phi$

(d) $\mathsf{A}\,\mathsf{G}\,\mathsf{A}\,\mathsf{F}\,q$ — $\mathcal{M}, s_0 \not\models \phi,\ \mathcal{M}, s_2 \not\models \phi$

Consider the paths from each of the states that end up looping in s_1.

Question 5.

5.1) (b) $\mathsf{E}\,\mathsf{F}(\phi \vee \psi) \equiv \mathsf{E}\,\mathsf{F}\phi \vee \mathsf{E}\,\mathsf{F}\psi$ — Equivalent, proved earlier.

(c) $\mathsf{A}\,\mathsf{G}(\phi \vee \psi) \equiv \mathsf{A}\,\mathsf{G}\phi \vee \mathsf{E}\,\mathsf{F}\psi$ — Equivalent.

(g) $\top \equiv \mathsf{A}\,\mathsf{G}\phi \Rightarrow \mathsf{E}\,\mathsf{G}\phi$ — Equivalent.

(h) $\top \equiv \mathsf{E}\,\mathsf{G}\phi \Rightarrow \mathsf{A}\,\mathsf{G}\phi$ — Not equivalent. Consider the model in Figure 2 and take the initial state to be s_1, with $\phi = r$. The self-loop causes $\mathsf{E}\,\mathsf{G}\phi$ to hold, but on no other path does $\mathsf{G}\phi$ hold, so the implication breaks.

5.2) (a) $\mathsf{A}\,\mathsf{G}(\phi \wedge \psi) \equiv \mathsf{A}\,\mathsf{G}\phi ? \mathsf{A}\,\mathsf{G}\psi$ — $? \rightarrow \wedge$

(b) $\mathsf{E}\,\mathsf{F}\neg\phi \equiv \neg ?? \phi$ — $?? \rightarrow \mathsf{A}\,\mathsf{G}$

Question 6.

I used the following code to encode the system

```
1
2  -- NOT looped
3  MODULE x1
4
5  VAR
6      state: {on, off};
7
8  ASSIGN
9      init(state) := off;
10     next(state) :=
11         case
12             out : off;
13             !out: on;
14         esac;
15
16 DEFINE
17     out := (state = on);
18
19 -- XOR looped
20 MODULE x2(inp)
21
22 VAR
23     state: {on, off};
24
25 ASSIGN
26     init(state) := off;
27     next(state) :=
28         case
29              inp &  out : off;
30              inp & !out : on;
31             !inp &  out : on;
32             !inp & !out : off;
33         esac;
34
```

```
DEFINE
    out := (state = on);

MODULE main

VAR
    m0: x1();
    m1: x2(m0.out);

SPEC

AG (m0.out <-> AX (! m0.out))
```

And received the following output from simulation (the specification was added after simulation)

```
▲ assignments/as1/code NuSMV -int q6.smv
... LICENSE TRIMMED

NuSMV > go
NuSMV > pick_state -r
NuSMV > print_current_state -v
Current state is 1.1
m0.state = off
m1.state = off
NuSMV > simulate -r -k 5
********  Simulation Starting From State 1.1   ********
NuSMV > show_traces -t
There is 1 trace currently available.
NuSMV > show_traces -v
    <!-- ################## Trace number: 1 ################## -->
Trace Description: Simulation Trace
Trace Type: Simulation
  -> State: 1.1 <-
    m0.state = off
    m1.state = off
    m0.out = FALSE
    m1.out = FALSE
  -> State: 1.2 <-
    m0.state = on
    m1.state = off
    m0.out = TRUE
    m1.out = FALSE
  -> State: 1.3 <-
    m0.state = off
    m1.state = on
    m0.out = FALSE
    m1.out = TRUE
  -> State: 1.4 <-
    m0.state = on
```

```
    m1.state = on
    m0.out = TRUE
    m1.out = TRUE
  -> State: 1.5 <-
    m0.state = off
    m1.state = off
    m0.out = FALSE
    m1.out = FALSE
  -> State: 1.6 <-
    m0.state = on
    m1.state = off
    m0.out = TRUE
    m1.out = FALSE
NuSMV >
```

The required property holds, as checked in another run of the same program in batch mode.

```
SPEC
AG (m0.out <-> AX (! m0.out))
```

Question 7.

The file was downloaded from Piazza. The mutex condition was verified to hold as written in the example. The nospurious condition was coded as follows and found to hold as well.

```
INVARSPEC
(
(e1.ack-out -> e1.Request)
&(e2.ack-out -> e2.Request)
&(e3.ack-out -> e3.Request)
&(e4.ack-out -> e4.Request)
&(e5.ack-out -> e5.Request)
)
```

The property such that if a request is held on, it is eventually acknowledged, is given by

$$\mathsf{G}((\mathsf{G}\,\mathsf{req}_i) \Rightarrow \mathsf{F}\,\mathsf{ack}_i) \tag{1}$$

It was coded in as

```
LTLSPEC
    G(
    ((G e1.Request) -> (F e1.ack-out))
    &((G e3.Request) -> (F e3.ack-out))
    )
```

However, NuSMV verified this specification as being true for the model.

The no loss invariant was coded in

```
INVARSPEC
    (
       (!e1.loss)
      &(!e2.loss)
      &(!e3.loss)
      &(!e4.loss)
      &(!e5.loss)
    )
```

and NuSMV found it to be false.

The 3 consecutive loss property is stated as

$$\mathsf{A}\,\mathsf{G}\neg(\mathtt{loss} \wedge \mathsf{A}\,\mathsf{X}\,(\mathtt{loss} \wedge \mathsf{A}\,\mathsf{X}\,\mathtt{loss})) \tag{2}$$

and was coded as

```
CTLSPEC
    (
       (AG (! (e1.loss & AX(e1.loss & AX(e1.loss)))))
      &(AG (! (e2.loss & AX(e2.loss & AX(e2.loss)))))
      &(AG (! (e3.loss & AX(e3.loss & AX(e3.loss)))))
      &(AG (! (e4.loss & AX(e4.loss & AX(e4.loss)))))
      &(AG (! (e5.loss & AX(e5.loss & AX(e5.loss)))))
    )
```

with NuSMV once again verifying it to be true.

Finally modifying the circuit to use the value of incoming token instead of last token value

```
    next(Persistent) := Request & (Persistent | Token);

    -- CHANGED TO -->

    next(Persistent) := Request & (Persistent | token-in);
```

However, curiously, running this, NuSMV reported **no change in specification validity.**

Appendix

Final modified code (using the modified model) with specifications used

```
--syncarb.smv:

MODULE arbiter-element(above,below,init-token)

VAR
  Persistent : boolean;
  Token : boolean;
  Request : boolean;

ASSIGN
  init(Token) := init-token;
  next(Token) := token-in;
  init(Persistent) := FALSE;
  next(Persistent) := Request & (Persistent | token-in);

DEFINE
  above.token-in := Token;
  override-out := above.override-out | (Persistent & Token);
  grant-out := !Request & below.grant-out;
  ack-out := Request & (Persistent & Token | below.grant-out);
  loss := Request & !ack-out;

MODULE main

VAR
  e5 : arbiter-element(self,e4,FALSE);
  e4 : arbiter-element(e5,e3,FALSE);
  e3 : arbiter-element(e4,e2,FALSE);
  e2 : arbiter-element(e3,e1,FALSE);
  e1 : arbiter-element(e2,self,TRUE);

DEFINE
  grant-in := TRUE;
  e1.token-in := token-in;
  override-out := FALSE;
  grant-out := grant-in & !e1.override-out;

--MUTEX
INVARSPEC
    (
     !(e1.ack-out & e2.ack-out)

     & !(e1.ack-out & e3.ack-out)
     & !(e2.ack-out & e3.ack-out)

     & !(e1.ack-out & e4.ack-out)
     & !(e2.ack-out & e4.ack-out)
     & !(e3.ack-out & e4.ack-out)

     & !(e1.ack-out & e5.ack-out)
     & !(e2.ack-out & e5.ack-out)
     & !(e3.ack-out & e5.ack-out)
     & !(e4.ack-out & e5.ack-out)
    )

```

```
INVARSPEC
    (
       (e1.ack-out -> e1.Request)
      &(e2.ack-out -> e2.Request)
      &(e3.ack-out -> e3.Request)
      &(e4.ack-out -> e4.Request)
      &(e5.ack-out -> e5.Request)
    )

LTLSPEC
    G(
    ((G e1.Request) -> (F e1.ack-out))
    &((G e3.Request) -> (F e3.ack-out))
    )

INVARSPEC
    (
       (!e1.loss)
      &(!e2.loss)
      &(!e3.loss)
      &(!e4.loss)
      &(!e5.loss)
    )

CTLSPEC
    (
       (AG (! (e1.loss & AX(e1.loss & AX(e1.loss)))))
      &(AG (! (e2.loss & AX(e2.loss & AX(e2.loss)))))
      &(AG (! (e3.loss & AX(e3.loss & AX(e3.loss)))))
      &(AG (! (e4.loss & AX(e4.loss & AX(e4.loss)))))
      &(AG (! (e5.loss & AX(e5.loss & AX(e5.loss)))))
    )
```