CIS 4930/6930: Principles of Cyber-Physical Systems

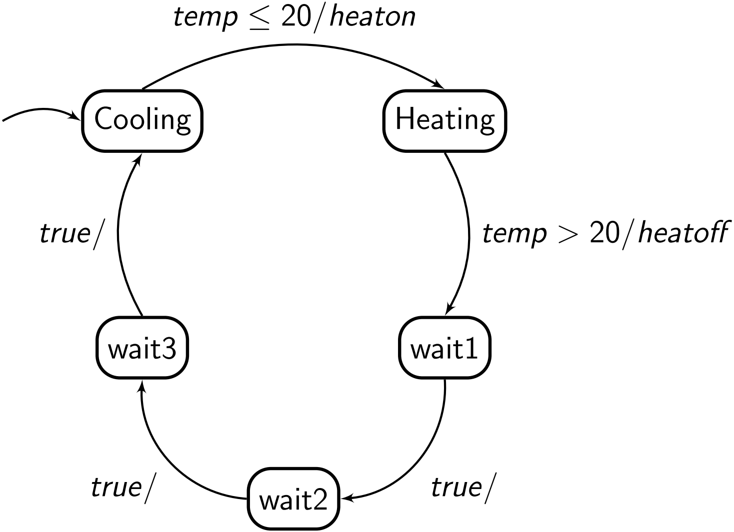
Homework Solutions

Hao Zheng

Department of Computer Science and Engineering
University of South Florida

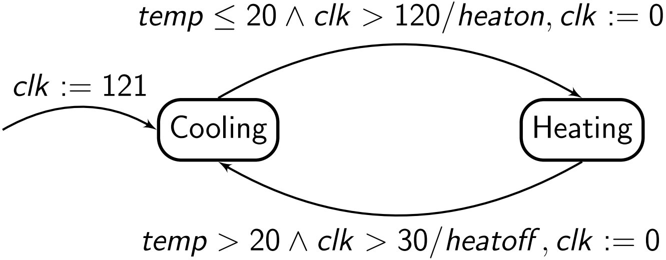
HW 2

HW 2: Chapter 3, Problem 2

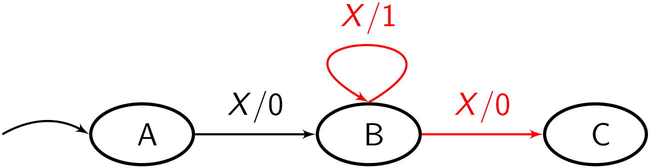


HW 2: Chapter 3, Problem 2

input: temp, clk : $\mathbb{R}$
outputs: heaton, heatoff: pure



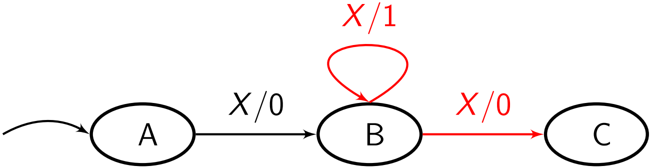
HW 2: Chapter 3, Problem 5



Problem 5:

- **b** $x = (p, p, p, p, p, \ldots), \quad y = (0, 1, 1, 0, a, \ldots)$ **Yes**
- **c** $x = (a, p, a, p, a, \ldots), \quad y = (a, 1, a, 0, a, \ldots)$ **No**
- **d** $x = (p, p, p, p, p, \ldots), \quad y = (0, 0, a, a, a, \ldots)$ **Yes**
- **e** $x = (p, p, p, p, p, \ldots), \quad y = (0, a, 0, a, a, \ldots)$ **No**

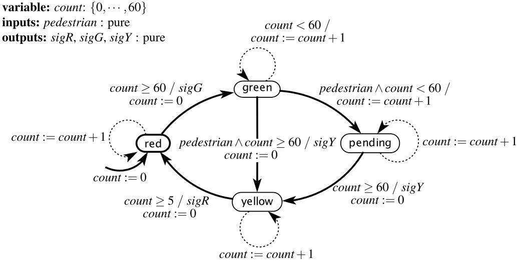
HW 2: Chapter 3, Problem 5



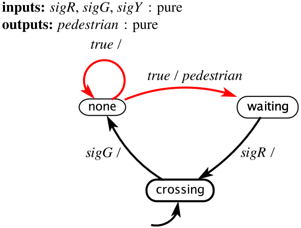
Problem 5:

- **a** $x = (p, p, p, p, p, \ldots)$, $y = (0, 1, 1, 0, 0, \ldots)$ **No**
- **b** $x = (p, p, p, p, p, \ldots)$, $y = (0, 1, 1, 0, a, \ldots)$ **Yes**
- **c** $x = (a, p, a, p, a, \ldots)$, $y = (a, 1, a, 0, a, \ldots)$ **No**
- **d** $x = (p, p, p, p, p, \ldots)$, $y = (0, 0, a, a, a, \ldots)$ **Yes**
- **e** $x = (p, p, p, p, p, \ldots)$, $y = (0, a, 0, a, a, \ldots)$ **No**

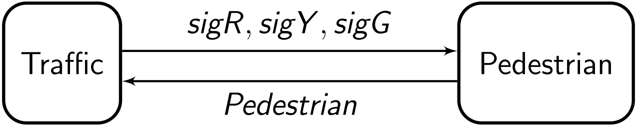
HW 2: Problem 3



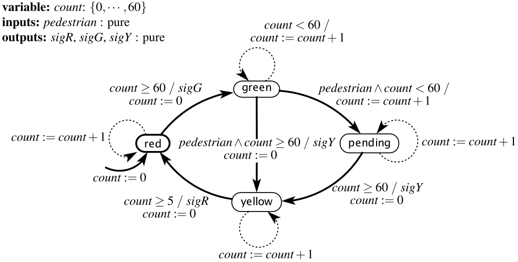
variable: $count$: $\{0, \cdots, 60\}$
inputs: $pedestrian$: pure
outputs: $sigR$, $sigG$, $sigY$: pure

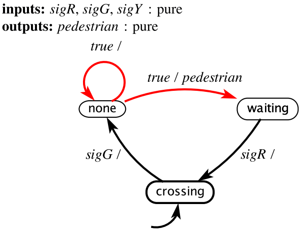


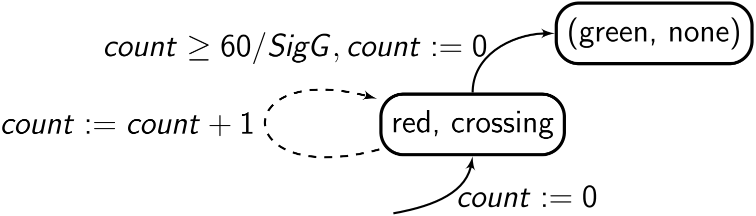
inputs: $sigR$, $sigG$, $sigY$: pure
outputs: $pedestrian$: pure



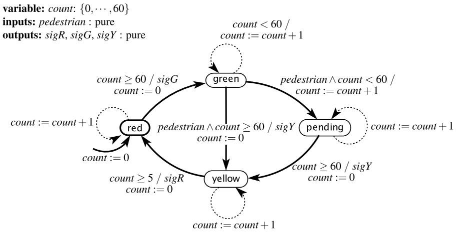
HW 2: Problem 3

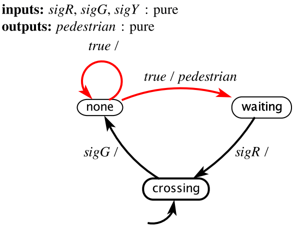


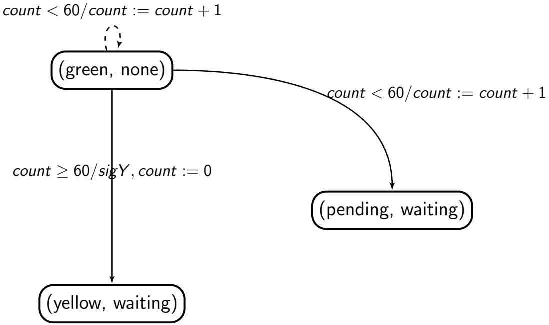




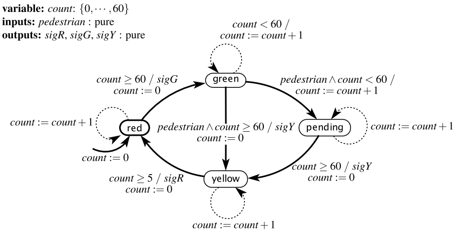
HW 2: Problem 3

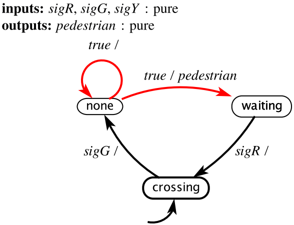


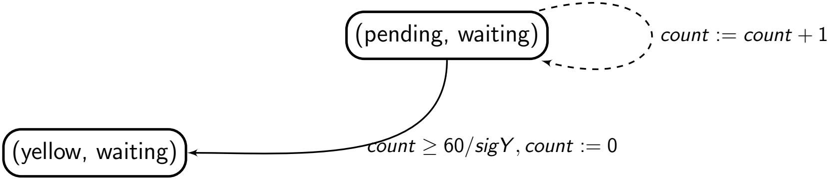




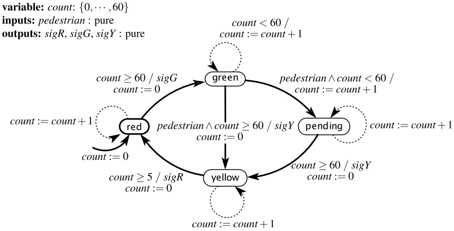
HW 2: Problem 3

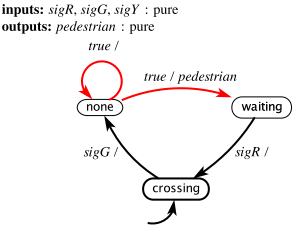


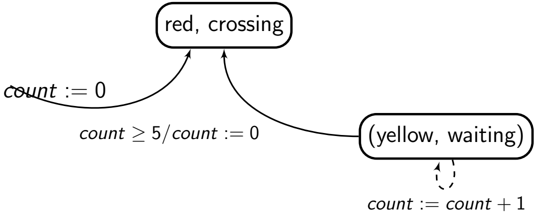




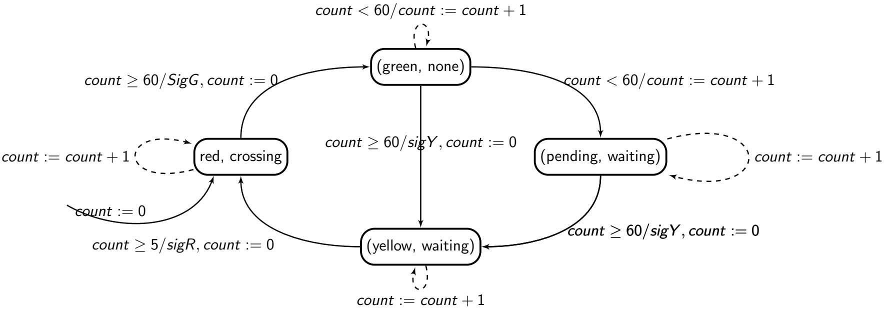
HW 2: Problem 3





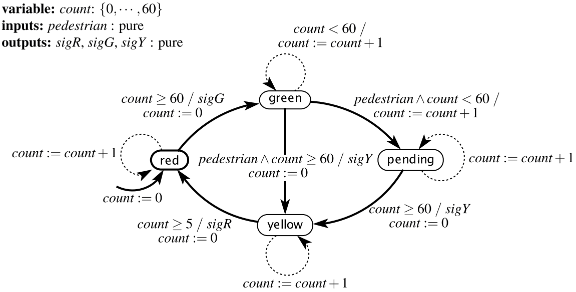


HW 2: Problem 3



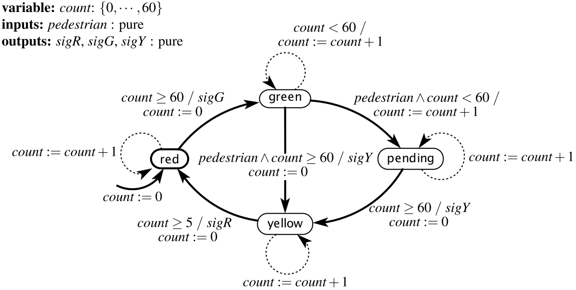
HW 3: Pointers and Hints

HW 3, Problem 1



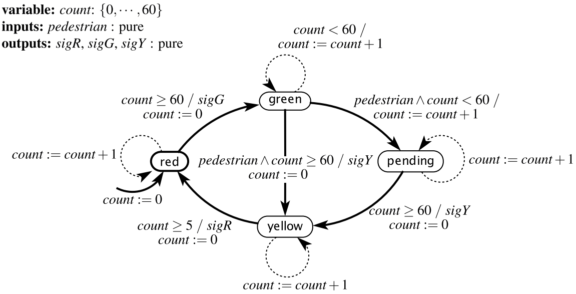
```
mtype = {sigR, sigY, sigG};
chan signal = [0] of {mtype};

chan ped = [0] of {bit};
```

HW 3, Problem 1



```
active proctype traffic() {
 ...
Green:
  if  /*  Not allowed by SPIN */
  :: ped?1 && count < 60  -> ...; goto Pending;
  :: ped?1 && count >= 60 -> ...; goto Yellow;
  :: count < 60           -> ...; goto Green;
```

HW 3, Problem 1



```
active proctype traffic() {
 ...
Green: ped?pedBit;  /* Lead to invalid end state! */
  if
  :: pedBit==1 && count < 60  -> ...; goto Pending;
  :: pedBit==1 && count >= 60 -> ...; goto Yellow;
  :: count < 60              -> count++; goto Green;
```

HW 3, Problem 1

```
/* Also lead to invalid end state! */
active proctype traffic() {
 ...
Green:
  if
  :: ped?1 -> count < 60  -> ...; goto Pending;
  :: ped?1 -> count >= 60 -> ...; goto Yellow;
  :: count < 60              -> ...; goto Green;
  ...
```

HW 3, Problem 1

```
/* Still lead to invalid end state! */
active proctype traffic() {
 ...
Green:
  if
  :: ped?1 -> if
              :: count < 60  -> count++; goto Pending;
              :: count >= 60 -> signal!sigY;
                                count=0;
                                goto Yellow;
              fi
  :: count < 60            -> ...; goto Green;
  ...
```

HW 3, Problem 1

```
/* Finally ... */
active proctype traffic() {
 ...
Green:
  if
  :: ped?1 -> if
              :: count < 60  -> count++; goto Pending;
              :: count >= 60 -> count=0;
                                goto Yellow;
              fi
  :: count < 60            -> ...; goto Green;
  ...
```

HW 3, Problem 1, Check Properties

Property #1

Pedestrians are allowed to cross the street only when the traffic light is red,

```
bool traffic_red = false;

active proctype traffic() {
  Red:     ...  traffic_red = false; goto Green; ...
  Green:   ...
  Pending: ...
  Yellow: ... traffic_red = true; goto red ...   }

active proctype traffic() {  ...   }
```

HW 3, Problem 1, Check Properties

Property #1

Pedestrians are allowed to cross the street only when the traffic light is red,

```
bool traffic_red = false;
bool ped_cross = false;

active proctype traffic() {  ...   }

active proctype pedestrian() {
  Crossing: signal?sigG -> ped_cross = false; goto None
  None:  ped!1 -> ped_cross = false; goto Waiting;
  Waiting: signal?sigR -> ped_cross = true; goto Crossi
}
```

HW 3, Problem 1, Check Properties

Property #1

Pedestrians are allowed to cross the street only when the traffic light is red,

```
bool traffic_red = false;
bool ped_cross = false;

active proctype traffic() {  ...   }
active proctype pedestrian() {  ...  }

active proctype monitor() { /* Prop \#1 is checked */
   assert( ped_cross -> traffic_red );
}
```

HW 3: Post-Submission Discussioins

HW 3, Problem 1

```
mtype = {sigR, sigY, sigG};
chan signal = [0] of {mtype};
chan ped = [0] of {bit};
int count;
bool traffic_red = true;  /*  The initial state of traffic light is
bool ped_cross = true; /* The initial state of pedestrian light is
bool ped_pres = false;

active proctype traffic()
{
red:
if
:: atomic { count >= 60 -> signal!sigG; count = 0;
                           traffic_red = false; goto green; }
:: atomic { else ->  count++; traffic_red = true; goto red; }
fi;
...
}
```

HW 3, Problem 1, How SPIN Works?

```
active proctype traffic()
{
red:
if
:: atomic { count >= 60 -> signal!sigG; count = 0;
                           traffic_red = false; goto green; }
:: atomic { else ->  count++; traffic_red = true; goto red; }
fi;
...
}

active proctype pedestrian()
{  crossing:  atomic { signal?sigG -> ped_pres = false;
                                      ped_cross = false; goto none; }
...  }

active proctype monitor()
{  assert(!ped_cross || traffic_red);  }
```

HW 3, Problem 1, How SPIN Works?

```
active proctype traffic()
{
red:
if
:: atomic { count >= 60 -> signal!sigG; count = 0;
                           traffic_red = false; goto green; }
:: atomic { else ->  count++; traffic_red = true; goto red; }
fi;
...
}

active proctype pedestrian()
{  crossing:  atomic { signal?sigG -> ped_pres = false;
                                     ped_cross = false; goto none; }
...  }

ltl prop1 { [](!ped_cross || traffic_red) }
ltl prop2 { [](ped_pres -> <> ped_cross) }
```

HW 3, Problem 1, Another Version

```
#define RED 1
#define YELLOW 2
#define GREEN 3
#define PENDING 4

#define CROSSING 5
#define NONE 6
#define WAITING 7

byte traffic_state = RED;
byte ped_state = CROSSING;
```

HW 3, Problem 1, Another Version

```
active proctype traffic()
{
do
:: traffic_state==RED ->
     if
     :: atomic { count >= 60 -> signal!GREEN; count = 0;
                              traffic_state = GREEN; }
     :: atomic { else ->  count++; traffic_state = RED; }
     fi;

:: traffic_state==GREEN ->...

:: traffic_state == PENDING -> ...

:: traffic_state == YELLOW -> ...
od
}
```

HW 3, Problem 1, Another Version

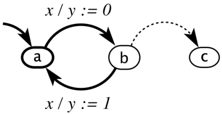
```
active proctype pedestrian()
{ do
  :: ped_state == CROSSING ->
                    atomic { signal?GREEN -> ped_state = NONE; }

  :: ped_state == NONE -> atomic { ped!1 -> ped_state = WAITING; }

  :: ped_state == WAITING ->
                    atomic { signal?RED -> ped_state = CROSSING; }
  od  }

ltl prop1 { []((ped_state != CROSSING) || (traffic_state==RED)) }
ltl prop2 { [](ped_state == WAITING -> <> (ped_state == CROSSING)) }
```

HW 4

Problem 1

1. Consider the following state machine:

input: x: pure
output: y: $\{0,1\}$



(Recall that the dashed line represents a default transition.) For each of the following LTL formulas, determine whether it is true or false, and if it is false, give a counterexample:

(a) $x \implies \mathbf{F}\mathsf{b}$ true

(b) $\mathbf{G}(x \implies \mathbf{F}(y=1))$ false: $a \xrightarrow{x} b \xrightarrow{\neg x} c \xrightarrow{x} c \ldots$

(c) $(\mathbf{G}x) \implies \mathbf{F}(y=1)$ true

(d) $(\mathbf{G}x) \implies \mathbf{GF}(y=1)$ true

(e) $\mathbf{G}((\mathsf{b} \wedge \neg x) \implies \mathbf{FG}\mathsf{c})$ true

(f) $\mathbf{G}((\mathsf{b} \wedge \neg x) \implies \mathbf{G}\mathsf{c})$ false: b and c are different states.

(g) $(\mathbf{GF}\neg x) \implies \mathbf{FG}\mathsf{c}$ false: $a \xrightarrow{x} b \xrightarrow{x} a \xrightarrow{\neg x} a \xrightarrow{x} b \xrightarrow{x} a \xrightarrow{\neg x} a \ldots$

Problem 2

2. This problem is concerned with specifying in linear temporal logic tasks to be performed by a robot. Suppose the robot must visit a set of n locations $l_1, l_2, \ldots, l_n$. Let p_i be an atomic formula that is *true* if and only if the robot visits location l_i.

   Give LTL formulas specifying the following tasks:

   (a) The robot must eventually visit at least one of the n locations.

   (b) The robot must eventually visit all n locations, but in any order.

   (c) The robot must eventually visit all n locations, in the order $l_1, l_2, \ldots, l_n$.

(a) $\vee_{1 \leq i \leq n} \mathbf{F} p_i$

(b) $\wedge_{1 \leq i \leq n} \mathbf{F} p_i$

(c) $\mathbf{F}(p_1 \wedge \mathbf{F}(P_2 \wedge \mathbf{F}(p_3 \wedge \ldots)))$

$\mathbf{F}(p_1 \wedge \mathbf{XF}(P_2 \wedge \mathbf{XF}(p_3 \wedge \ldots)))$ is the same as in (c). Why?

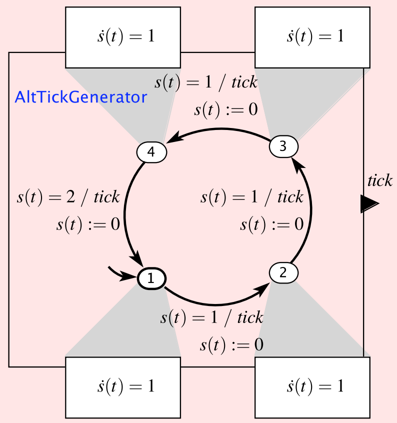
HW 5

Question 1

Construct (on paper is sufficient) a timed automaton similar to that of Figure 4.7 which produces *tick* at times $1, 2, 3, 5, 6, 7, 8, 10, 11, \cdots$. That is, ticks are produced with intervals between them of 1 second (three times) and 2 seconds (once).

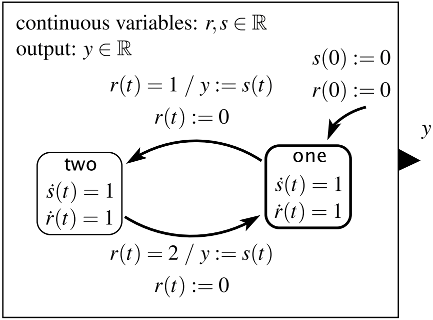
Question 1

Construct (on paper is sufficient) a timed automaton similar to that of Figure 4.7 which produces *tick* at times $1, 2, 3, 5, 6, 7, 8, 10, 11, \cdots$. That is, ticks are produced with intervals between them of 1 second (three times) and 2 seconds (once).



Question 2(a)

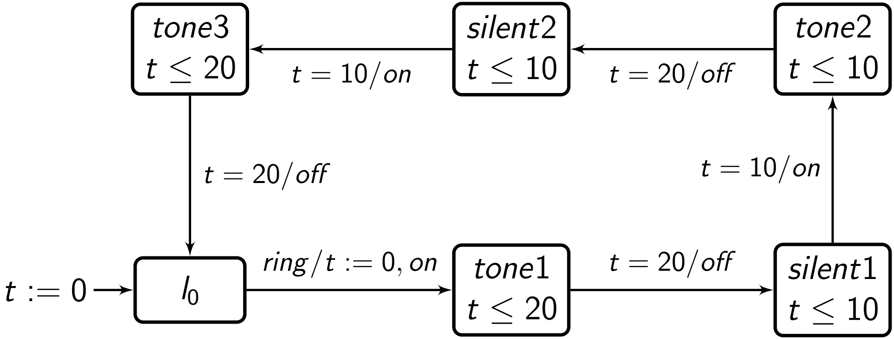
For the timed automaton shown below, describe the output y. Avoid imprecise or sloppy notation.



Question 3

You have an analog source that produces a pure tone. You can switch the source on or off by the input event *on* or *off*. Construct a timed automaton that provides the *on* and *off* signals as outputs, to be connected to the inputs of the tone generator. Your system should behave as follows. Upon receiving an input event *ring*, it should produce an 80 ms-long sound consisting of three 20 ms-long bursts of the pure tone separated by two 10 ms intervals of silence. What does your system do if it receives two *ring* events that are 50 ms apart?

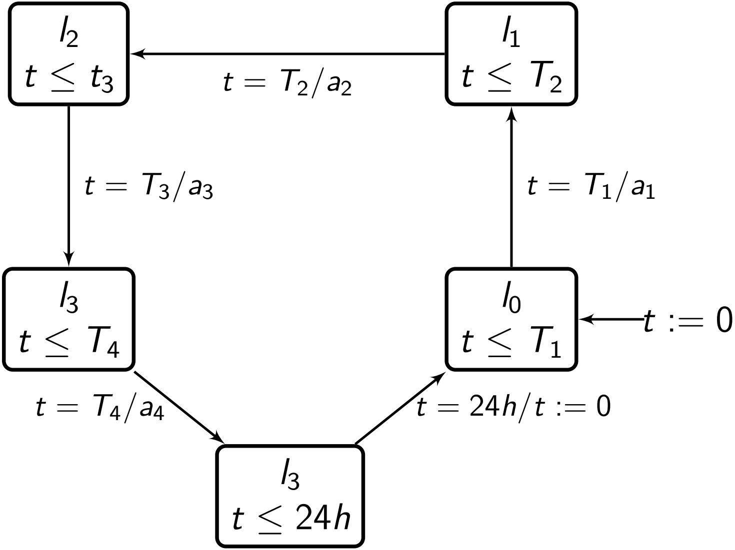
Question 3


$tone3$
$t \leq 20$
$t = 10/on$
$silent2$
$t \leq 10$
$t = 20/off$
$tone2$
$t \leq 10$
$t = 20/off$
$t = 10/on$
$t := 0$
l_0
$ring/t := 0, on$
$tone1$
$t \leq 20$
$t = 20/off$
$silent1$
$t \leq 10$

Question 5

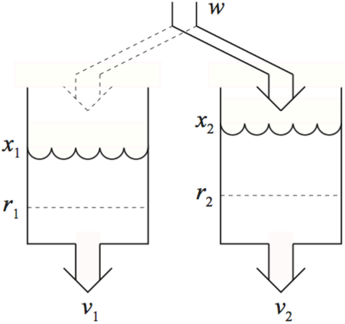
A programmable thermostat allows you to select 4 times, $0 \leq T_1 \leq \cdots \leq T_4 < 24$ (for a 24-hour cycle) and the corresponding setpoint temperatures $a_1, \cdots, a_4$. Construct a timed automaton that sends the event a_i to the heating systems controller. The controller maintains the temperature close to the value a_i until it receives the next event. How many timers and modes do you need?

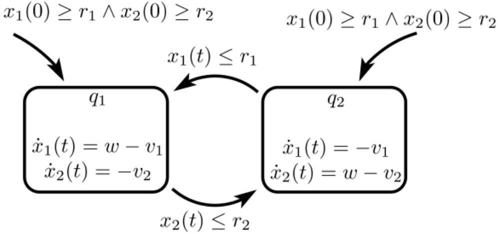
Question 5


l_2
$t \leq t_3$
l_1
$t \leq T_2$
$t = T_2/a_2$
$t = T_3/a_3$
$t = T_1/a_1$
l_3
$t \leq T_4$
l_0
$t \leq T_1$
$t := 0$
$t = T_4/a_4$
$t = 24h/t := 0$
l_3
$t \leq 24h$

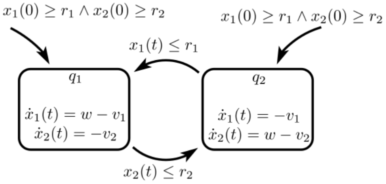
HW 7

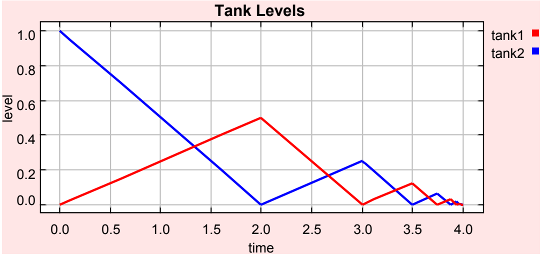
Water Tank


w
x_1
r_1
x_2
r_2
v_1
v_2


$x_1(0) \geq r_1 \wedge x_2(0) \geq r_2$
$x_1(0) \geq r_1 \wedge x_2(0) \geq r_2$
$x_1(t) \leq r_1$
q_1
$\dot{x}_1(t) = w - v_1$
$\dot{x}_2(t) = -v_2$
q_2
$\dot{x}_1(t) = -v_1$
$\dot{x}_2(t) = w - v_2$
$x_2(t) \leq r_2$

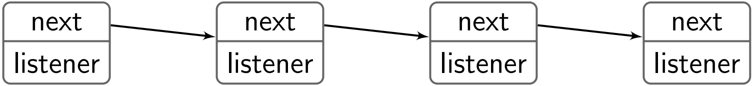
Water Tank Trajectory





Problem P2

- To show that concurrent access to a list may corrupt the data structure.
- Modeling a linked list?



Use arrays in Promela.

```
bool next[4];
bool listener[4];
int tail = -1;
int head = -1;
```

Problem P2

```
proctype addListener()
{
   if
   :: head==-1 -> head = 0;
                  next[head]=true;
                  listener[head] = true;
                  tail = head;
   :: else -> tail = tail+1;
              next[tail] = true;
              listener[tail] = true;
   fi
}
```

Problem P2

```
init
{
  next[0] = false;  next[1] = false;
  next[2] = false;  next[3] = false;
  listener[0] = false;  listener[1] = false;
  listener[2] = false;  listener[3] = false;

  atomic { run addListener(); run addListener();
           run addListener(); run addListener(); };

  (_nr_pr==1) ->
            assert(listener[0] && ... && listener[3]);
}
```