Homework 1

CS6501-006: Safety and Security in CPS

William Young

March 2, 2017

1. **Synchronous Models**

   Possible reactions:

$$0 \xrightarrow{0/(0,0)} 0$$

$$0 \xrightarrow{1/(0,0)} 1$$

$$0 \xrightarrow{1/(0,1)} 1$$

$$1 \xrightarrow{0/(1,0)} 0$$

$$1 \xrightarrow{0/(1,1)} 0$$

$$1 \xrightarrow{1/(1,1)} 1$$

   From $z := \mathit{choose}(x, u)$, we conclude that z awaits x. However, y is independent of x $(u := 0, y := u)$ and therefore does not await x.

2. **Safety Requirements**

   The property $\phi : x \geq 0$ is an inductive invariant of the transition system T if

   (a) Every initial state of T satisfies ϕ

   (b) If a state satisfies ϕ and (s, t) is a transition of T, then t must satisfy ϕ.

   We first consider initial state $(x = 0)$. This satisfies ϕ; that is, $x \geq 0$, satisfying (a).

   We then consider an arbitrary state $s \mid s(x) = a$.

   Assume that s satisfies ϕ; that is, assume $a \geq 0$.

   Consider the state $t(x) = a - 1$ obtained by executing a transition from s. We must show that t satisfies ϕ.

   Consider the case $a = 0$, $t(x) = a - 1$. In order for t to satisfy ϕ, it must be the case that $a \geq 0$, but we have $a = -1$. Since t fails to satisfy the property ϕ, ϕ is not an inductive invariant.

Next, consider the property ψ that strengthens ϕ:

$$\begin{aligned} \phi_1 &: \text{mode}=\{\text{off}\} \rightarrow x \geq 0 \\ \& \ \phi_2 &: \text{mode}=\{\text{on}\} \rightarrow x > 0 \end{aligned}$$

We observe that ψ implies ϕ; that is, both properties maintain $x \geq 0$. Now we shall prove ψ is an inductive invariant using proof by induction.

Base case:

Consider initial state (mode=$\{\text{off}\}, x = 0$). This satisfies ψ; that is, (mode=$\{\text{off}\} \rightarrow x \geq 0$).

Inductive case:

Consider an arbitrary state s with $x = a$ and mode $= b$.

Assume that s satisfies ψ; that is, assume

$$\begin{aligned} \phi_1 &: b = \{\text{off}\} \rightarrow a \geq 0 \\ \& \ \phi_2 &: b = \{\text{on}\} \rightarrow a > 0 \end{aligned}$$

Consider the state t obtained by executing a transition from s.

If $t(\text{mode}) = \text{off}$ then $t(x) = a + 1$ and $t(\text{mode}) \in \{\text{on, off}\}$.
From our assumption $a \geq 0$, it follows that $a + 1 > 0$ and $b \in$\{on,off\}, satisfying ϕ_1 and ϕ_2.

If $t(\text{mode}) = \text{on}$ then $t(x) = a - 1$.
From our assumption $a > 0$,it follows that $a - 1 \geq 0$ and $b \in$\{on,off\}, satisfying ϕ_1 and ϕ_2.

In either case, the condition

$$\begin{aligned} \phi_1 &: t(\text{mode}) = \{\text{off}\} \rightarrow t(x) \geq 0 \\ \& \ \phi_2 &: t(\text{mode}) = \{\text{on}\} \rightarrow t(x) > 0 \end{aligned}$$

holds, therefore property ψ is an inductive invariant.

3. **Asynchronous Models**

   The asynchronous model `AsyncAdd` consists of the following elements (model illustrated on next page):

   (a) Input set: `nat` $\{x_1, x_2\}$
   (b) Output set: `nat` $\{y\}$
   (c) State variable set: `queue(nat)` $\{x1, x2\}$
   (d) Initial state: $x1 =$ `null`, $x2 =$ `null`

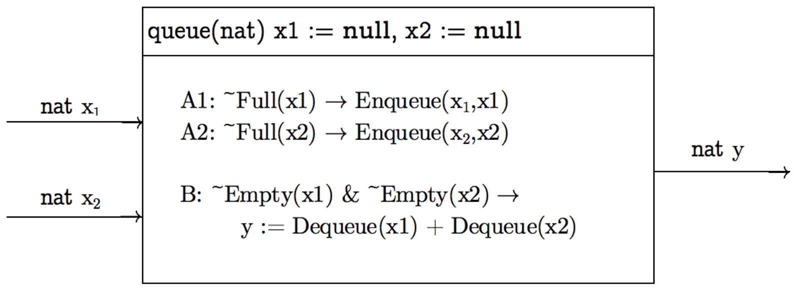

Figure 1: Asynchronous Adder Process.

(e) Input tasks:

- A1: $\neg\text{Full}(x1) \rightarrow \text{Enqueue}(x_1, x1)$
- A2: $\neg\text{Full}(x2) \rightarrow \text{Enqueue}(x_2, x2)$

(f) Output tasks:

- B: $\neg\text{Empty}(x1) \& \neg\text{Empty}(x2) \rightarrow y := \text{Dequeue}(x1) + \text{Dequeue}(x2)$

4. **Liveness Requirements**

(a) *The first is stronger than the second.*

Reasoning:

Suppose a trace ρ satisfies `Eventually`$(\phi_1 \;\&\; \phi_2)$.

There exists a position j such that $(\rho, j) \models \phi_1 \;\&\; \phi_2$.

It follows that both $(\rho, j) \models \phi_1$ and $(\rho, j) \models \phi_2$.

Since $(\rho, j) \models \phi_1$ it also satisfies `Eventually`(ϕ_1). Similarly, it also satisfies `Eventually`(ϕ_2).

It follows that the trace satisfies `Eventually`(ϕ_1) & `Eventually`(ϕ_2)

However, the two are not equivalent. Consider the trace $\rho = \{0, 1, 2, 3, 4\}$ over a boolean variable x. It satisfies `Eventually`$(x = 2)$ and `Eventually`$(x = 4)$, but does not satisfy `Eventually`$(x = 2 \;\&\; x = 4)$.

(b) *The two are equivalent.*

Reasoning:

Suppose a trace ρ satisfies `Eventually`$(\phi_1 \mid \phi_2)$.

There exists a position j such that $(\rho, j) \models \phi_1 \mid \phi_2$.

In other words, either $(\rho, j) \models \phi_1$ or $(\rho, j) \models \phi_2$.

Suppose $(\rho, j) \models \phi_1$. Then ρ satisfies `Eventually`$(\phi_1 \mid \phi_2)$.

Hence ρ also satisfies `Eventually`$(\phi_1) \mid$ `Eventually`(ϕ_2).

Now consider the converse.

Suppose a trace ρ satisfies `Eventually`(ϕ_1) | `Eventually`(ϕ_2).
Specifically, suppose it satisfies `Eventually`(ϕ_1).
There exists a position j such that $(\rho, j) \models \phi_1$.
It follows that $(\rho, j) \models \phi_1 \mid \phi_2$.
Thus, ρ satisfies `Eventually`$(\phi_1 \mid \phi_2)$.

(c) *The first is stronger than the second.*
Reasoning:
Suppose a trace ρ satisfies `Always Eventually`$(\phi_1 \ \& \ \phi_2)$.
For every position j, $(\rho, j) \models$ `Eventually`$(\phi_1 \ \& \ \phi_2)$.
For every j, there exists a position $i \geq j$ such that $(\rho, i) \models (\phi_1 \ \& \ \phi_2)$.
In other words, for every j, there exists a position $i \geq j$ such that both $(\rho, i) \models \phi_1$ and $(\rho, i) \models \phi_2$.
Since for every j, there exists a position $i \geq j$ such that $(\rho, i) \models \phi_1$, it satisfies `Always Eventually`(ϕ_1). Similarly, it satisfies `Always Eventually`(ϕ_2).
It follows that the trace satisfies `Always Eventually`(ϕ_1) & `Always Eventually`(ϕ_2).
However, the two are not equivalent. Consider the trace $\rho = \{0, 1, 0, 1, 0, 1...\}$ over a boolean variable x. It satisifies `Always Eventually`$(x = 0)$ & `Always Eventually`$(x = 1)$, but does not satisfy `Always Eventually`$(x = 0 \ \& \ x = 1)$.

(d) *The two are equivalent.*
Reasoning:
Suppose a trace ρ satisfies `Always Eventually`$(\phi_1 \mid \phi_2)$.
For every position j, $(\rho, j) \models$ `Eventually`$(\phi_1 \mid \phi_2)$.
For every j, there exists a position $i \geq j$ such that $(\rho, i) \models (\phi_1 \mid \phi_2)$.
In other words, for every j, there exists a position $i \geq j$ such that either $(\rho, i) \models \phi_1$ or $(\rho, i) \models \phi_2$.
Suppose $(\rho, i) \models \phi_1$. Then ρ satisfies `Always Eventually`$(\phi_1 \mid \phi_2)$.
Hence ρ also satisfies `Always Eventually`(ϕ_1) | `Always Eventually`(ϕ_2).
Now consider the converse.
Suppose a trace ρ satisfies `Always Eventually`(ϕ_1) | `Always Eventually`(ϕ_2).
Specifically, suppose it satisfies `Always Eventually`(ϕ_1).
For every j, there exists a position $i \geq j$ such that $(\rho, i) \models \phi_1$.
It follows that $(\rho, i) \models \phi_1 \mid \phi_2$.
Thus, ρ satisfies `Always Eventually`$(\phi_1 \mid \phi_2)$.

5. **Timed Models**

I used a custom python script, dbm.py (next page), to generate the following DBMs. The script required manual setup for each DBM in order to set the bounds and zero-outs/guards/clock invariants. In particular, I had to set the input and guard/invariant DBMs and specify whether a clock variable reset occurred. The code uses the principles of intersection and the canonicalization algorithm to produce canonicalized DBMs as output.

- $R_A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
- $R'_A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix}$
- $R_B = \begin{bmatrix} 0 & -2 & 0 & -2 \\ 3 & 0 & 3 & 0 \\ 0 & -2 & 0 & -2 \\ 3 & 0 & 3 & 0 \end{bmatrix}$
- $R'_B = \begin{bmatrix} 0 & -2 & 0 & -2 \\ 5 & 0 & 3 & 0 \\ 2 & -2 & 0 & -2 \\ 5 & 0 & 3 & 0 \end{bmatrix}$
- $R_C = \begin{bmatrix} 0 & -3 & 0 & 0 \\ 5 & 0 & 3 & 5 \\ 2 & -2 & 0 & 2 \\ 0 & -3 & 0 & 0 \end{bmatrix}$
- $R'_C = \begin{bmatrix} 0 & -3 & 0 & 0 \\ 8 & 0 & 3 & 5 \\ 6 & -2 & 0 & 2 \\ 5 & -3 & 0 & 0 \end{bmatrix}$

dbm.py

```
#!/usr/local/bin/python3
# setup ###################################################
import sys
import math
inf = math.inf
zero = int(sys.argv[1]) # set to 1 if a variable is being zeroed
arg = int(sys.argv[2])  # provide the variable (1, 2, 3) being zeroed
# DBMs ####################################################
e1 = [[  0, -2,  0, -2],[  5,  0,  3,  0],[  2, -2,  0, -2],[  5,  0,  3,  0]]
e2 = [[  0, -3,inf,inf],[inf,  0,inf,inf],[inf,inf,  0,inf],[inf,inf,inf,  0]]
e3 = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
# intersect ###############################################
def intersect(e1, e2, e3):
    for i in range(0,4):
        for j in range(0,4):
            if e1[i][j] <= e2[i][j]:
                e3[i][j] = e1[i][j]
            else:
                e3[i][j] = e2[i][j]
# canonicalize ############################################
def canonicalize(e3):
    for l in range(0,4):
        for i in range(0,4):
            for j in range(0,4):
                e3[i][j] = min(e3[i][j],e3[i][l]+e3[l][j])
# main ####################################################
intersect(e1, e2, e3)
canonicalize(e3)

if zero == 1:
    e3[0][arg] = 0
    e3[arg][0] = 0
    ps = [1,2,3]
    ps.remove(arg)
    for index in ps:
        e3[arg][index] = e3[0][index]
        e3[index][arg] = e3[index][0]
    canonicalize(e3)
# results #################################################
print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in e3]))
print( str(-1 * e3[0][1]) + ' <= x1 <= ' + str(e3[1][0]) )
print( str(-1 * e3[0][2]) + ' <= x2 <= ' + str(e3[2][0]) )
print( str(-1 * e3[0][3]) + ' <= x3 <= ' + str(e3[3][0]) )
print( str(-1 * e3[2][1]) + ' <= x1 - x2 <= ' + str(e3[1][2]) )
print( str(-1 * e3[3][2]) + ' <= x2 - x3 <= ' + str(e3[2][3]) )
print( str(-1 * e3[3][1]) + ' <= x1 - x3 <= ' + str(e3[1][3]) )
```