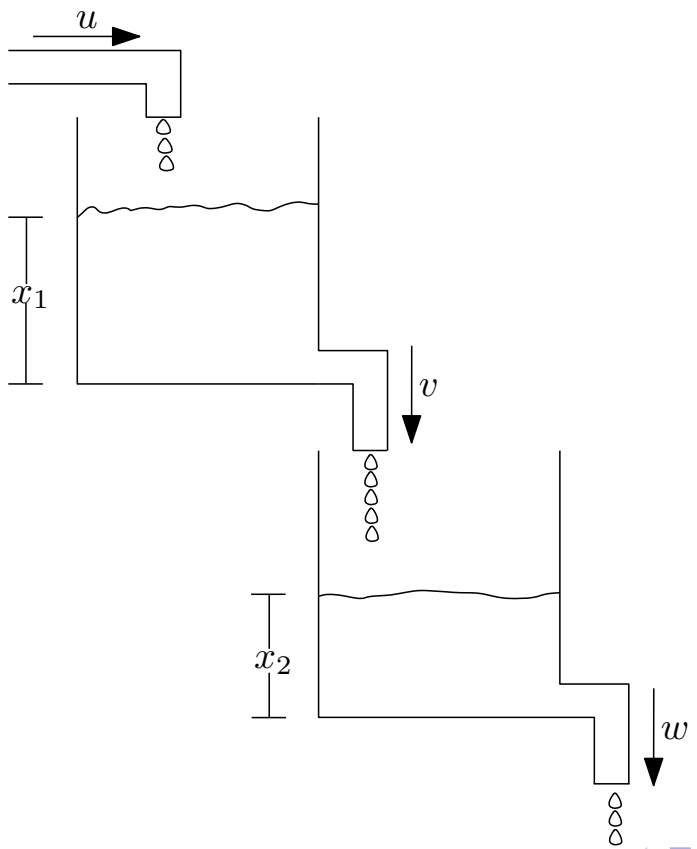
Formal Systems
Tutorial
Hybrid System Modeling, Program Analysis

Department of Computer Science & Engineering,
Indian Institute of Technology, Kharagpur

1. There are three taps in the system, namely Tap-1 having a flow rate of $u = 5$, Tap-2 having a flow capacity of $v = 2$, and Tap-3 having a flow capacity of $w = 4$. Tap-2 and Tap-3 are always on. Tap-1 is switched on when $x_1 + x_2$ falls below 10 and is switched off when x_1 exceeds 80. Initially, we have $x_1 = 50$ and $x_2 = 50$. Draw a hybrid automaton for the system. Explain the dynamics of the system.



2. Hot-Air Balloons: Some interesting facts...

Modern hot air balloons, with an onboard heat source, were developed by Ed Yost. Hot air balloons are able to fly to extremely high altitudes. The burner unit gasifies liquid propane, mixes it with air, ignites the mixture, and directs the flame and exhaust into the mouth of the envelope.

Draw a Hybrid Automata model for the Hot-air balloon described below.

The Hot-air balloon has a gas tank that when full, has a volume of 10 units. When the burner is burning the gas, the hot air balloon rises at a rate of $\frac{t^2}{2}$, where t is the amount of time for which the burner has been continuously on. When the burner is off, the hot-air balloon rises at a rate of $-t^2$, t being the amount of time for which the burner has been off.

When the hot-air-balloon runs out of fuel, the fuel tank may be re-placed. Re-placing the fuel tank takes 3 units of time. During this time the balloon will be falling. The hot-air balloon drains at a rate of 1 unit per unit time.

You need to design a strategy that ensures that the Hot-air balloon stays between heights of 100 and 120 units; after having first crossed 100 units of height.

3. Consider the following program P in a C like language.

```
L0:  a = b = i = 0;
L1:  while (a <= 1000) {
L2:      a = b + i;
L3:      b = a + 1;
L4:      i = i + 1;
L5:  }
L6:  if (b > 2000) { error: exit(-1);
L7:  }
```

3.1 Construct a Boolean program corresponding to this program P, using only the predicates $(a \leq 1000)$, $(b \geq 0)$, $(i = 0)$, and $(b \leq 2000)$

3.2 Show that the error location is reachable in the Boolean program you constructed

3.3 Explain whether the error identified in this Boolean program is a spurious counter-example.

4. Compute the Weakest Precondition for the following code snippets, given the postcondition:

4.1 Code Snippet 1:

```
a := 2*(b-1)-1
{a>0}
```

4.2 Code Snippet 2:

```
a := a+2*b-1
{a>1}
```

4.3 Code Snippet 3:

```
a := 2*b+1
b := a-3
{b < 0}
```

5. Analyze the program using the domain Parity and then Sign.

```
1.  y:=5; x:=-2*y;
2.  if(x>0){
3.        x:=x-(y%2-1);
4.        y:=x*(y-1);
5.  }
6.  else y:=-1;
```

5.4 Code Snippet 4:

```
a := 3*(2*b+a)
b := 2*a-1
{b>5}
```

5.5 Code Snippet 5:

```
if(a==b)
    b=2*a+1;
else
    b=2*a;
{b>1}
```

6. Compute the weakest precondition for the assignment statement $a := 2 * (b - 1) - 1$ given the postcondition $\{a > 0)$. Substitute the right hand side of the assignment statement for a in the postcondition
   $2 * (b - 1) > 0$
   $2b - 2 > 0$
   $2b > 2$
   $b > 1$

7. Compute the weakest precondition for the following sequence of assignment statements, for the postcondition given.
   $a := 2 * b + 1;$
   $b := a - 3$
   $b < 0$

   Substitute the right hand side of the second assignment in the postcondition to get a postcondition for the first assignment, then substitute the right hand side of the first assignment in that postcondition to get the precondition.
   $a - 3 < 0$
   $a < 3$ weakest precondition for first assignment
   $2 * b + 1 < 3$
   $2b < 2$
   $b < 1$

8. Consider the following code fragment:

```
1.  lock();
2.  gotLock = 1;
3.  while(*) {
4.        if(*) {
5.              if(gotLock == 1) {
6.                    unlock();
7.                    gotLock = 0;
                }
        }
8.      if(gotLock!=1) {
9.              lock();
10.             gotLock = 1;
        }
    }
11. unlock();
12.
```

8.1 Draw the initial control flow graph (CFG), up to the point where the first error is found (in which case other parts of the CFG may still be incomplete), or else explores the entire search space and finds no error. In your CFG:

- Number the nodes to match the number of the line that's about to execute.
- Label each node with "L" or "U" to represent the "lock==1" or "lock==0" predicates, respectively (the initial node should be labeled "U" for unlocked/lock==0).
- Label each non-branching edge with the statement executed.
- For branches, label P(pred) where pred is the condition that must be true for that branch to be taken (thus if there are two edges one will be pred and one !pred). For example, P(gotLock==1) will go on one edge in the CFG. You need not use any label if the predicate is *.

8.2 If an error state is reachable in the CFG given above, does it represent a real bug? If so, prove it is a real bug, by showing using Hoare Logic that the precondition of the path is not false. If the bug is not real, show using Hoare Logic that the precondition of the path is false.

8.3 Now, build the CFG again, this time considering the predicate you found.

9. Consider the following code fragment:

```
1:  if(*){
7:        do{
                    gotLock = 0;
8:                  if(*) {
9:                            lock();
                              gotLock++;
                    }
10:                 if(gotLock){
11:                           unlock();
                    }
12:       } while (*);
    }
2:  do {
          lock();
          old = new
3:        if(*) {
4:                  unlock();
                    new++;
          }
5:  } while (new!=old);
6:  unlock();
    return;}
```

8. Contd...

   8.1 Use predicate Lock=1 and Lock=0 to analyse the program for the **Double Lock** or the **Double Unlock** error.
   8.2 On finding an error (if at all), give a proof for the error.
   8.3 If an error is found to be spurious, which predicate should be added to improve the Abstraction. Re-analyze the program for possible errors.