MIDTERM EXAM

Models of Software Systems
Instructor: Dawn McLaughlin

October 19, 2018

IMPORTANT NOTES:

- This exam should have 13 pages, including this cover sheet. If it does not, please notify your instructor(s) immediately.
- You may use your books and notes for this exam, but YOU MAY NOT DISCUSS THIS EXAM WITH ANYONE. If you have questions about the exam, feel free to ask me privately by e-mail.
- Exams should be submitted via Canvas by 7:00AM Eastern (Daylight) time on Monday, October 22, 2018.

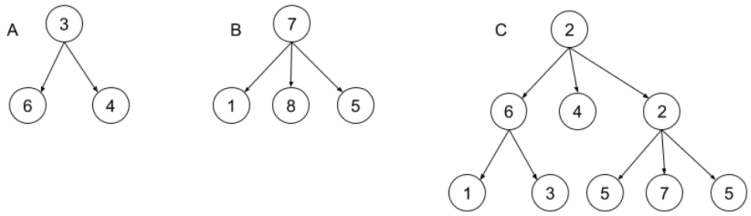
PART 1:

Consider the following definition of trees that expands on the variety already seen on homework assignments:

$$TREE ::= leaf\langle\langle\mathbb{N}\rangle\rangle | binary\_node\langle\langle\mathbb{N}\times TREE\times TREE\rangle\rangle | ternary\_node\langle\langle\mathbb{N}\times TREE\times TREE\times TREE\rangle\rangle$$

It says that a tree is either a leaf which has a numeric value, it is a binary node which has a numeric value and has two subtrees, or it is a ternary node which has a numeric value and has three subtrees.

Here are some examples of trees:



Tree A has two leaves and one binary node; tree B has three leaves and one ternary node; tree C has six leaves, one binary node, and two ternary nodes.

Tree A can be described as

$$binary\_node(3, leaf(6), leaf(4))$$

and tree B as

$$ternary\_node(7, leaf(1), leaf(8), leaf(5))$$

.

1. Give the description of tree C in the same format. [3 points]

$ternary\_node(2, binary\_node(6, leaf(1), leaf(3)), leaf(4), ternary\_node(2, leaf(5), leaf(7), leaf(5)))$

Now consider the following function definitions:

$$
\begin{array}{l}
leaves : TREE \to \mathbb{N} \\
nodes : TREE \to \mathbb{N} \\
weight : TREE \to \mathbb{N} \\
\hline
\forall t_1, t_2, t_3 : TREE \bullet \forall n : \mathbb{N} \bullet \\
\quad leaves(leaf(n)) = 1 \land \\
\quad leaves(binary\_node(n, t_1, t_2)) = leaves(t_1) + leaves(t_2) \land \\
\quad leaves(ternary\_node(n, t_1, t_2, t_3)) = leaves(t_1) + leaves(t_2) + leaves(t_3) \land \\
\\
\quad nodes(leaf(n)) = 0 \land \\
\quad nodes(binary\_node(n, t_1, t_2)) = 1 + nodes(t_1) + nodes(t_2) \land \\
\quad nodes(ternary\_node(n, t_1, t_2, t_3)) = 1 + nodes(t_1) + nodes(t_2) + nodes(t_3) \land \\
\\
\quad weight(leaf(n)) = n \land \\
\quad weight(binary\_node(n, t_1, t_2)) = n + weight(t_1) + weight(t_2) \land \\
\quad weight(ternary\_node(n, t_1, t_2, t_3)) = n + weight(t_1) + weight(t_2) + weight(t_3)
\end{array}
$$

2. The above definition of $nodes$ counts all nodes in the tree. Write axiomatic definitions for functions $binary\_nodes$ and $ternary\_nodes$ that count only the binary and ternary nodes of the tree, respectively. You may give one axiomatic definition for both functions, as was done for $leaves$, $nodes$, and $weight$, above, or you can do two separate axiomatic definitions, as you prefer. [2×5 = 10 points]

   I merely took the nodes axiomatic definition and tweaked it to only add binary nodes in the $binary\_nodes$ definition and ternary nodes in the $ternary\_nodes$ definition.

$$
\begin{array}{l}
binary\_nodes : TREE \to \mathbb{N} \\
\hline
\forall t_1, t_2, t_3 : TREE \bullet \forall n : \mathbb{N} \bullet \\
\quad binary\_nodes(leaf(n)) = 0 \land \\
\quad binary\_nodes(binary\_node(n, t_1, t_2)) = 1 + binary\_nodes(t_1) + binary\_nodes(t_2) \land \\
\quad binary\_nodes(ternary\_node(n, t_1, t_2, t_3)) = binary\_nodes(t_1) + binary\_nodes(t_2) \\
+ binary\_nodes(t_3)
\end{array}
$$

$$ternary\_nodes : TREE \to \mathbb{N}$$

$$\forall t_1, t_2, t_3 : TREE \bullet \forall n : \mathbb{N} \bullet$$

$$ternary\_nodes(leaf(n)) = 0 \land$$
$$ternary\_nodes(binary\_node(n, t_1, t_2)) = ternary\_nodes(t_1) + ternary\_nodes(t_2) \land$$
$$ternary\_nodes(ternary\_node(n, t_1, t_2, t_3)) = 1 + ternary\_nodes(t_1) + ternary\_nodes(t_2) + ternary\_nodes(t_3)$$

3. Consider the following sets and predicates about these trees:

$TREE$	the set of all trees
$three\_heavy(t)$	t is $three\_heavy$ if and only if it has strictly more ternary nodes than binary nodes (it must have at least one ternary node to be $three\_heavy$)
$two\_heavy(t)$	t is two_heavy if and only if it has strictly more binary nodes than ternary nodes (it must have at least one binary node to be two_heavy)
$balanced(t)$	t is $balanced$ if and only if it has the same number of binary and ternary nodes (including none of either)

Using these predicates and the functions defined above, translate the following sentences about trees into predicate logic (note that some of the sentences may be false according to the definitions of the predicates). You may use standard arithmetic relations and functions (such as $<$, $>$, $\max$, $\min$, etc.), as required:

a. [3 points] A tree is $three\_heavy$ if and only if it is a leaf or has more ternary nodes than binary nodes.
$\forall t : TREE \bullet (is\_leaf(t) \lor ternary\_nodes(t) > binary\_nodes(t)) \Rightarrow is\_tree\_heavy(t)$

b. [3 points] Every $three\_heavy$ tree $weigh$s more than some two_heavy tree.

c. [3 points] Some $three\_heavy$ tree $weigh$s more than every two_heavy tree.
$\exists t : THREE\_HEAVY \bullet \forall w : TWO\_HEAVY \bullet weight(t) > weight(w)$

d. [3 points] The immediate subtrees of any $balanced$ tree are either all $balanced$ or all have the same $weight$.
$\forall t : BALANCED\_TREES \bullet \Rightarrow balanced(t.subrees) \lor same\_weight(t.subtrees)$

4. Prove the following by structural induction: [15 points]

$$\forall t : TREE \bullet nodes(t) = binary\_nodes(t) + ternary\_nodes(t)$$

Base Case: Show that the property holds for leaf, that is $nodes(leaf(t)) = binary\_nodes(leaf(t)) + ternary\_nodes(leaf(t))$

nodes(leaf(t))	
=	[Applying the definition of nodes(leaf(t))]
0	
=	[Aritmetic]
0 + 0	
=	[Applying the definition of $binary\_nodes(leaf(t))$ and $ternary\_nodes(leaf(t))$]
$binary\_nodes(leaf(t)) + ternary\_nodes(leaf(t))$	

Induction case: assume that the property holds for trees t1, t2 and t3, that is, $nodes(t_1) = binary\_nodes(t_1) + ternary\_nodes(t_1)$, $nodes(t_2) = binary\_nodes(t_2) + ternary\_nodes(t_2)$ and $nodes(t_3) = binary\_nodes(t_3) + ternary\_nodes(t_3)$

nodes(node(t1, t2, t3))

= [definition of nodes]

$1 + nodes(t_1) + nodes(t_2) \wedge (1 + nodes(t_1)$
$+nodes(t_2) + nodes(t_3))$

= [Introduction hypotesis]

$(1 + binary\_nodes(t_1) + ternary\_nodes(t_1)$
$+binary\_nodes(t_2) + ternary\_nodes(t_2)) \wedge$
$(1 + binary\_nodes(t_1) + ternary\_nodes(t_1) +$
$binary\_nodes(t_2) + ternary\_nodes(t_2) +$
$binary\_nodes(t_3) + ternary\_nodes(t_3))$

= [Grouping all binary nodes and ternary nodes using the commutative property of +]

$(1 + binary\_nodes(t_1) + binary\_nodes(t_2) +$
$ternary\_nodes(t_1) + ternary\_nodes(t_2)) \wedge$
$(binary\_nodes(t_1) + binary\_nodes(t_2) +$
$binary\_nodes(t_3) + 1 + ternary\_nodes(t_1)$
$+ternary\_nodes(t_2) + ternary\_nodes(t_3))$

= [Substitution of definition of binary˙nodes and ternary nodes]

$(binary\_nodes(binary\_node(n, t_1, t_2)) +$
$ternary\_nodes(binary\_node(n, t_1, t_2))) \wedge$
$(binary\_nodes(ternary\_node(n, t_1, t_2, t_3)) +$
$ternary\_nodes(ternary\_node(n, t_1, t_2, t_3)))$

= [Factorizing $binary\_nodes$ and $ternary\_nodes$]

$binary\_nodes(node(t_1, t_2, t_3)) + ternary\_nodes(node(t_1, t_2, t_3))$

PART 2:

The following quadruple defines a state machine that models the (imaginary) Queequeg brand "AutoBarista Mini" coffee machine:

$$
\begin{aligned}
&\text{AutoBaristaMini} = \{\\
&\quad [espresso : \{1,2\}, milk : \{0,1\}],\\
&\quad \{s : S | espresso = 1 \wedge milk = 1\},\\
&\quad \{e\_button, m\_button, d\_button\},\\
&\quad \{(s,a,s') : S \times A \times S |\\
&\qquad a = e\_button \Rightarrow (s'(milk) = s(milk) \wedge s'(espresso) \neq s(espresso))\\
&\qquad \wedge\\
&\qquad a = m\_button \Rightarrow (s(milk) = 1 \wedge s'(milk) = 0 \wedge s'(espresso) = s(espresso))\\
&\qquad \wedge\\
&\qquad a = d\_button \Rightarrow (s'(milk) = 1 \wedge s'(espresso) = 1)\\
&\quad \}\\
&\qquad\qquad\qquad \}
\end{aligned}
$$

Answer each of the following questions with respect to the above model.

Definitions:

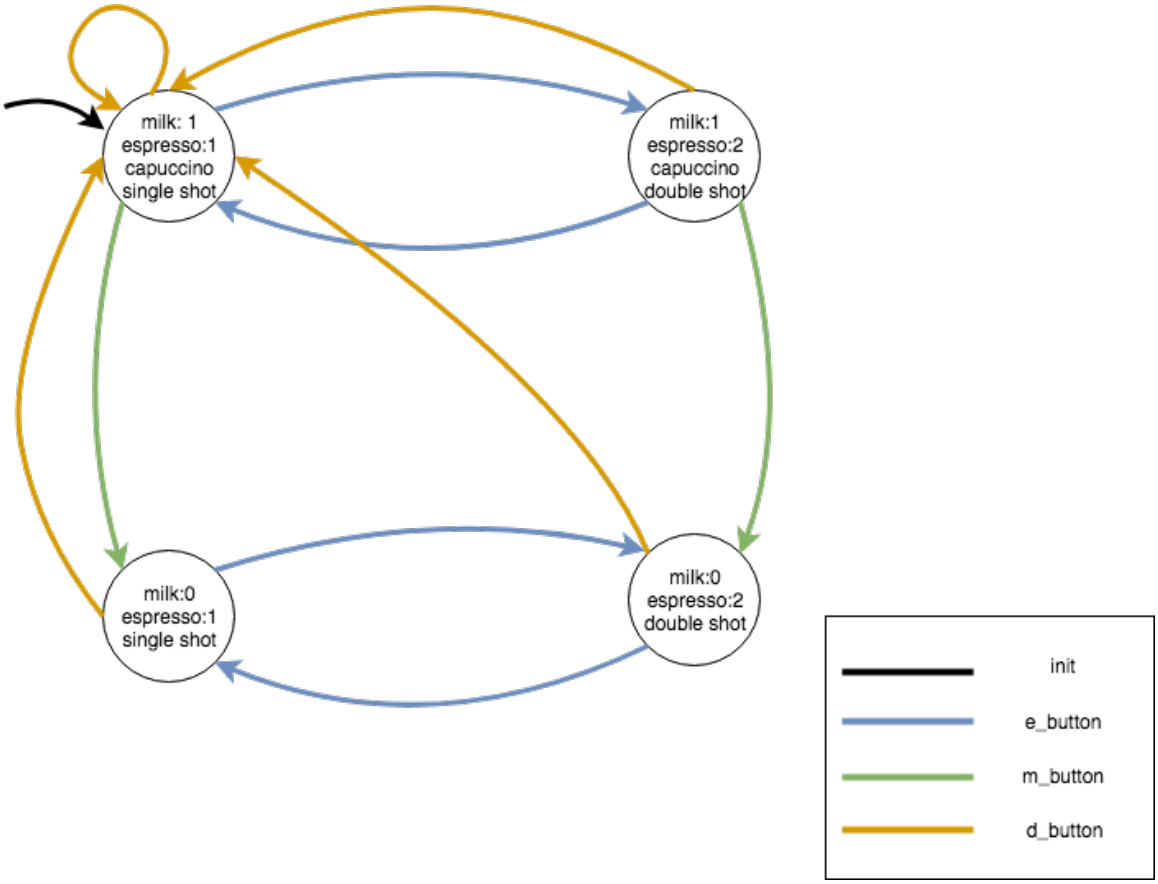
A cappuccino consists of steamed milk plus espresso.

The terms "single" and "double" refer to the number of shots of espresso a beverage contains.

1. Draw a transition diagram for AutoBaristaMini. You may name individual states if you wish. For each state, be sure to indicate clearly the values of all state variables. Don't forget to label transitions. [15 points]

   It took me a long time to figure out what the states meant in the machine, after reaching out a few times to Dawn and by looking at the other questions, the best I could come with is that the state variables represent the user selection from milk and number of espresso shots.

   To avoid cluttering the diagram with labels, I decided to color code the arrows, I could also have used arrows on both ends of the action line.



2. Do each of the following:

   (a) Rewrite the set δ by enumeration (i.e., list all the triples). You may use the names you assigned to states in question (1) if you wish. [5 points]

   $\delta = ($
   $\{capuccino\_single\_shot, e\_button, capuccino\_double\_shot\},$
   $\{capuccino\_single\_shot, m\_button, single\_shot\},$
   $\{capuccino\_single\_shot, d\_button, capuccino\_single\_shot\},$
   $\{capuccino\_double\_shot, m\_button, double\_shot\},$
   $\{capuccino\_double\_shot, d\_button, capuccino\_single\_shot\},$
   $\{capuccino\_double\_shot, e\_button, capuccino\_single\_shot\},$
   $\{double\_shot, d\_button, capuccino\_single\_shot\},$
   $\{double\_shot, e\_button, single\_shot\},$
   $\{single\_shot, d\_button, capuccino\_single\_shot\},$
   $\{single\_shot, e\_button, double\_shot\},$
   $)$

   (b) Give the pre- and post-conditions of each action. [3 $\times$ 5 = 15 points]

e_button makes espresso toggle between 1 and 2 shots and keeps milk selection the same.
$e\_button()$
pre:$true$
post: (espresso = 1 $\Rightarrow$ (espresso' = 2 $\wedge$ milk' = milk)) $\wedge$
(espresso = 2 $\Rightarrow$ (espresso' = 1 $\wedge$ milk' = milk))

m_button is only allowed if milk = 1, this threw me off a little bit but is what I was able to interpret from the state machine definition. It disables the milk.
$m\_button()$
pre:$milk = 1$
post : milk' = 0 $\wedge$ espresso' = espresso

$d\_button()$
pre:$true$
post: milk' = 1 $\wedge$ espresso' = 1

3. Answer each of the following: [5 $\times$ 1 = 5 points]

   (a) Assuming that the AutoBaristaMini supplies a coffee cup each time d_button is pressed and it dispenses the selected drink, is it possible to get an empty cup from the machine?

   Based on the provided allowed variable values for espresso and milk that we can extract from the states definition "$[espresso : \{1, 2\}, milk : \{0, 1\}]$" it is not possible because you have to get either 1 or 2 shots of espreseo, meaning, it is not possible to select 0 shots and only get milk or nothing.

   (b) Give an informal explanation of what the e_button action does, in terms that an actual user of the machine would understand, i.e., without referring to the formal model.

   Toggle the number of espresso shots between 1 and 2.

   (c) Give an informal explanation of what the m_button action does, in terms that an actual user of the machine would understand, i.e., without referring to the formal model.

   Disables milk, it is important to mention that it is not possible to enable it back without pressing d_button first.

   (d) How many different drinks can AutoBaristaMini make?

   4, $\{single\_shot, double\_shot, cappuccino\_single\_shot, capuccino\_double\_shot\}$ there's a state for each drink.

   (e) Give the event-based trace of the execution in which a double espresso, a single cappuccino, and a double cappuccino are dispensed, in that order.

   The question is asking for a single execution:
   $< m\_button, e\_button, d\_button, d\_button, e\_button, d\_button >$

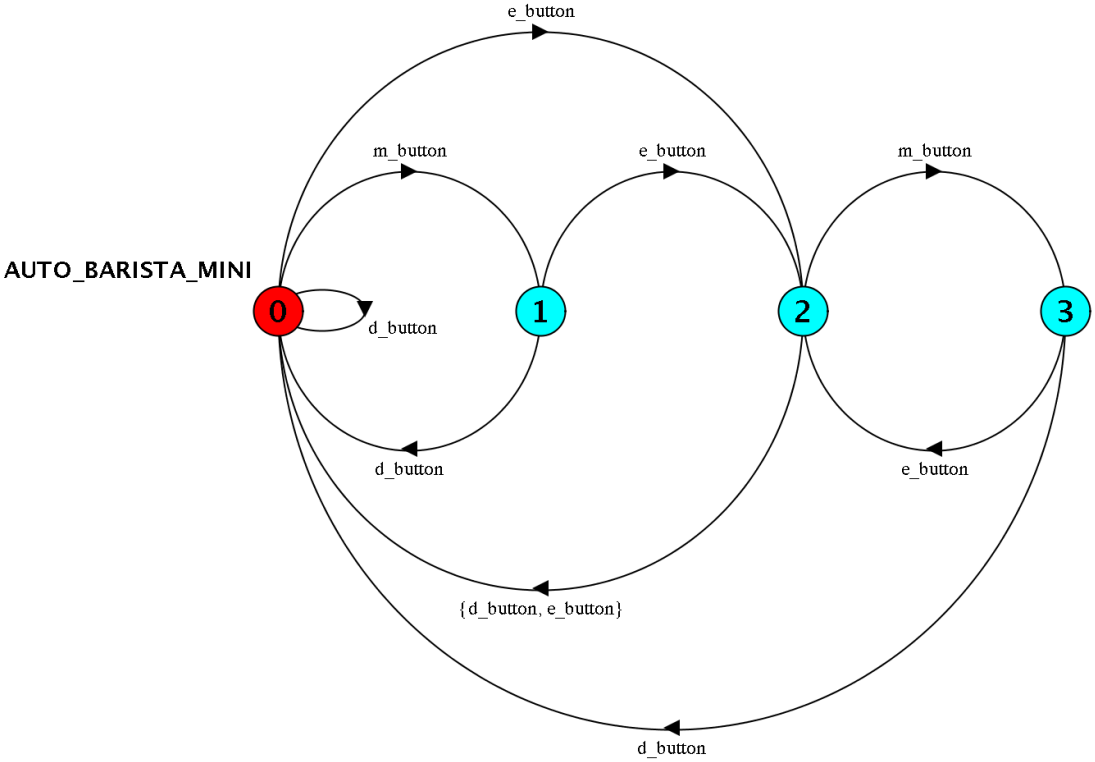
4. Give an FSP specification of AutoBaristaMini. [20 points]

```
//-----------------------------------------------------
//  AutoBarista Mini
//-----------------------------------------------------
//
//=====================
// Process Definitions
//=====================
//
// Autobarista starts in capuccino state
//
AUTO_BARISTA_MINI = CAPUCCINO_SINGLE_SHOT,
//
CAPUCCINO_SINGLE_SHOT = (
 e_button -> CAPUCCINO_DOUBLE_SHOT
 |m_button -> SINGLE_SHOT
 |d_button -> CAPUCCINO_SINGLE_SHOT
),

CAPUCCINO_DOUBLE_SHOT = (
e_button -> CAPUCCINO_SINGLE_SHOT
 |m_button -> DOUBLE_SHOT
 |d_button -> CAPUCCINO_SINGLE_SHOT
),

SINGLE_SHOT = (
  d_button -> CAPUCCINO_SINGLE_SHOT
 |e_button -> CAPUCCINO_DOUBLE_SHOT
),

DOUBLE_SHOT = (
  d_button -> CAPUCCINO_SINGLE_SHOT
 |e_button -> CAPUCCINO_DOUBLE_SHOT
).
```


AUTO_BARISTA_MINI
0
1
2
3
e_button
m_button
e_button
m_button
d_button
d_button
e_button
{d_button, e_button}
d_button

5. Consider the following additional behaviour of a more advanced model, the "AutoBarista Eco": The machine has a maximum stock of 100 cups. Each time it dispenses a drink, the user can put in their own reusable cup or the machine will automatically dispense a cup if no reusable cup is put in. When the machine is out of cups, it will not dispense a drink when no reusable cup is present. The act of adding more cups does not modify the drink that has been selected to be dispensed. The machine will always dispense a drink if a reusable cup has been inserted.

   Here are the updated specifications of S, I, and A for $\mathrm{AutoBaristaEco}$:

   $$S = [espresso : \{1, 2\}, milk : \{0, 1\}, cups : 0..100]$$
   $$I = \{s : S | espresso = 1 \wedge milk = 1 \wedge cups = 100\}$$
   $$A = \{e\_button, m\_button, d\_button\}$$

   The pre- and post-conditions for e_button and m_button for $\mathrm{AutoBaristaEco}$ differ from those for $\mathrm{AutoBaristaMini}$ only by adding $cups' = cups$ to the post-condition in each case.

   Give the pre- and post-conditions for the d_button action of $\mathrm{AutoBaristaEco}$.

   Model the reusable cup as an input: the input should be $true$ if a reusable cup has been inserted, $false$ if no cup has been inserted. [5 points]

   $BOOLEAN ::= true | false$
   $d\_button(has\_reusable\_cup : BOOLEAN)$
   pre: $has\_reusable\_cup = true \vee cups > 0$
   post: ($has\_reusable\_cup \Rightarrow$ (espresso' = 1 $\wedge$ milk' = 1 $\wedge$ cups' = cups)) $\wedge$
   ($cups > 0 \Rightarrow$ (espresso' = 1 $\wedge$ milk' = 1 $\wedge$ cups' = cups - 1))

PART 3:

The AutoBarista Mini and Eco are limited-functionality, lower-end models in the Queequeg line of coffee machines. Their basic model is the AutoBarista, which is currently in the process of being re-engineered.

The AutoBarista is capable of producing drinks containing 0, 1, or 2 shots of espresso, with or without steamed milk, and in drinks containing steamed milk, with or without a shot of flavour syrup (syrup is not available for drinks containing only espresso). The machine offers a choice of three different syrups. There are thus a total of 14 different drinks the machine can produce.

	espresso	milk	flavour
1	no_shot	yes_milk	$no\_flavour$
2	no_shot	yes_milk	$flavour\_1$
3	no_shot	yes_milk	$flavour\_2$
4	no_shot	yes_milk	$flavour\_3$
5	one_shot	yes_milk	$no\_flavour$
6	one_shot	yes_milk	$flavour\_1$
7	one_shot	yes_milk	$flavour\_2$
8	one_shot	yes_milk	$flavour\_3$
9	two_shots	yes_milk	$no\_flavour$
10	two_shots	yes_milk	$flavour\_1$
11	two_shots	yes_milk	$flavour\_2$
12	two_shots	yes_milk	$flavour\_3$
13	one_shot	no_milk	$no\_flavour$
14	two_shots	no_milk	$no\_flavour$

1. Complete the following state space schema for the AutoBarista by supplying an appropriate invariant: [5 points]

   $Espresso == \{no\_shot, one\_shot, two\_shots\}$
   $Milk == \{yes\_milk, no\_milk\}$
   $Flavour == \{flavour\_1, flavour\_2, flavour\_3, no\_flavour\}$

   The invariant that I chosed to use is that if no milk is selected then no flavour can be selected.

   AutoBarista

   $espresso : Espresso$
   $milk : Milk$
   $flavour : Flavour$

   $milk = no\_milk \Rightarrow flavour = no\_flavour$

2. Write an $InitAutoBarista$ schema that defines the initial state for the system. Explain why this initial state is consistent with the state space invariant given above. [5 points]

   Reasoning: The proposal is to init espresso to one shot and milk to yes, and no flavour, this is consistant with the invariant because we are respecting the rule: "beverage can contain a flavour only when milk has been selected".

$$
\begin{array}{|l}
\hline
InitAutoBarista \\
AutoBarista \\
\hline
espresso = one\_shot \\
milk = yes\_milk \\
flavour = no\_flavour \\
\hline
\end{array}
$$

3. Write an operation $ToggleMilk$ that changes the value of the milk selection. [5 points]
   Here I will propose a naive or not robust version of $ToggleMilk$ I'll present a safe version in the following question:
   One thing to notice is that there are no real inputs or outputs in this version:

$$
\begin{array}{|l}
\hline
ToggleMilk \\
\Delta AutoBarista \\
\hline
flavour = no\_flavour \\
(milk = yes\_milk \Rightarrow (milk' = no\_milk, flavour' = no\_flavour, espresso' = espresso) \\
\land milk = no\_milk \Rightarrow (milk' = yes\_milk, flavour' = no\_flavour, espresso' = espresso) \\
\hline
\end{array}
$$

4. If the $ToggleMilk$ operation is not robust, use the schema calculus to make a robust version of the operation (be sure to define any auxiliary schemata you use in the process); if the operation is robust, explain why that is the case. [5 points]
   Note: The presented robust version of $ToggleMilk$ assumes that the $SetFlavour$ operation is also robust and respects the invariant.

   We declare an schema to represent the Result of the $ToggleMilk$ operation:

   ToggleMilkReport ::= Ok | $Error\_flavour\_selected$

   Ok is used when the operation has been successful.

$$
\begin{array}{|l}
\hline
ToggleMilkSuccess \\
result! : ToggleMilkReport \\
\hline
result! : Ok \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
ToggleMilkError \\
\Delta AutoBarista \\
result! : ToggleMilkReport \\
\hline
flavour \neq no\_flavour) \\
result! : Error\_flavour\_selected \\
\hline
\end{array}
$$

   RToggleMilk $\hat{=}$ $(ToggleMilk \land ToggleMilkSuccess) \lor ToggleMilkError$