Homework #2 Solutions

CSE 403

Molly Brown

11 February 1997

Disclaimer: The solutions given here are by no means the ONLY correct solutions. If you don't agree with the answers here, let me know since they may be errors that were introduced in the typing process. –Molly

1. Petri Net Example 5.21

   (a) One sequence of transitions which cause *t1* and *t2* to starve: *t1*, *t3*, *t1*, *t3*, ...

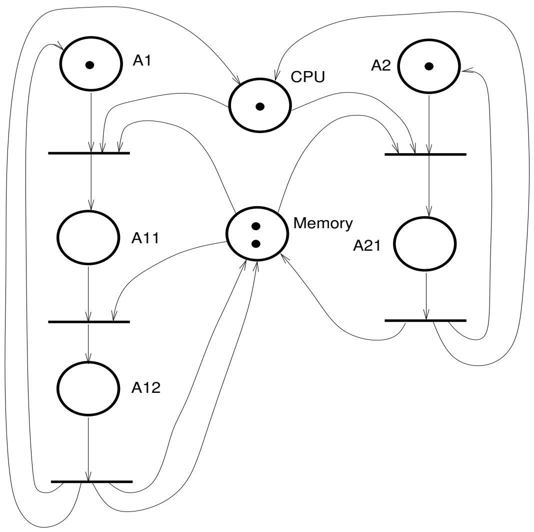
   (b) Once sequece fo transistions from which *t2* and *t4* can never become enabled again: *t2*, *t4*, *t2*

2. Drawing Petri Net

   Common mistakes:

   - Forgot to include places which signify the proceses are waiting to begin.
   - Only need one place initially marked with 2 tokens to represent the memory blocks.
   - Not returning 2 tokens to the Memory place when process A12 was completed.

   Example solution:



3. Real-time Logic representing the Alarm Clock Behavior

   Common mistakes:

   - Not ensuring that the first of the STOP button being pressed or 30 seconds passing would stop the Bell ringing.
   - Not including what event must occur to start the Bell ringing.

   Revised solution Prof. Shaw passed out in class:

$$\begin{aligned}\forall i \forall t_0((@(AC, i) = t_0) \quad \Rightarrow \quad & ((@(\uparrow Bell, i) = t_0) \wedge \\ & \exists t_1((@(\downarrow Bell, i) = t_1) \wedge \\ & (Press\_Stop \Rightarrow (@(STOP, j) = t_1)) \wedge \\ & (\neg Press\_Stop \Rightarrow (t_1 = t_0 + 30)))))\end{aligned}$$

$$\begin{aligned}\text{where Press\_Stop} \quad \equiv \quad & \exists j((t_0 < @(STOP, j)) \wedge (@(STOP, j) < t_0 + 30) \wedge \\ & \forall k((k < j) \Rightarrow (@(STOP, k) \leq t_0)))\end{aligned}$$

4. Real-time Logic Assertion for Traffic Lights

   Common mistakes:

   - Not putting the RTL statement in the form of an Assertion statement.
   - Not enforcing the relationship between the light cycle of the Street and the light cycle of the Avenue.

One solution (the simpliest one I could think of):

when *rs* {assert(@(rs,-2) < @(ga,-1) < @(ya,-1) < @(sa,-1) < @(gs,-1) < @(ys,-1) < @(rs,-1))}

5. Algebraic Specification for Unbounded Stack

   Common mistakes:

   - Omitting the domain and range for the added operations.
   - For Equal, not including cases where exactly one of the Stack parameters is NewStack.

$$\begin{array}{lrl} \text{Size:} & \text{Stack} & \rightarrow \text{Natural Number} \\ \text{Equal:} & \text{Stack} \times \text{Stack} & \rightarrow \text{Boolean} \end{array}$$

$$\begin{array}{lcl} \text{Size(NewStack)} & = & 0 \\ \text{Size(Push(}S_1\text{, i))} & = & \text{Size(}S_1\text{)} + 1 \\ \text{Equal(NewStack, NewStack)} & = & \text{TRUE} \\ \text{Equal(NewStack, Push(}S_1\text{, i))} & = & \text{FALSE} \\ \text{Equal(Push(}S_1\text{, i), NewStack)} & = & \text{FALSE} \\ \text{Equal(Push(}S_1, i_1\text{), Push(}S_2, i_2\text{)} & = & \text{if } i_1 = i_2 \\ & & \text{then Equal(}S_1, S_2\text{)} \\ & & \text{else FALSE} \end{array}$$

6. Algebraic Specification for Additional Set Operations

   Common mistakes:

   - Not specifying the domains and ranges for the added operations.
   - Implementing *Cardinality* as the number of items in the Set, not the number of *unique* items in the Set.
   - Not defining *Difference* and *Subset* for two sets that cannot be expressed in terms of the other.
   - There was some confusion about how the *Subset* operation was supposed to work. If you said explicitly how you interpreted the function, I graded it that way. Otherwise, I tried to figure out what intrepretation you were using and graded it accordingly.

$$\begin{array}{lrl} \text{Cardinality:} & \text{Set} & \rightarrow \text{Natural Number} \\ \text{Difference:} & \text{Set} \times \text{Set} & \rightarrow \text{Set} \\ \text{Subset:} & \text{Set} \times \text{Set} & \rightarrow \text{Boolean} \end{array}$$

$$
\begin{array}{lcl}
\text{Cardinality(NewSet)} & = & 0 \\
\text{Cardinality(Insert}(S,\, i)) & = & \text{if Member}(S,\, i) \\
 & & \text{then Cardinality}(S) \\
 & & \text{else Cardinality}(S) + 1 \\
\text{Difference(NewSet, NewSet)} & = & \text{NewSet} \\
\text{Difference(NewSet, } S_1) & = & \text{NewSet} \\
\text{Difference}(S_1, \text{ NewSet}) & = & S_1 \\
\text{Difference}(S_1, \text{ Insert}(S_2,\, i)) & = & \text{if Member}(S_1,\, i) \\
 & & \text{then Difference(Delete}(S_1,\, i), \text{ Delete}(S_2,\, i)) \\
 & & \text{else Difference}(S_1, \text{ Delete}(S_2,\, i)) \\
\text{Subset(NewSet, NewSet)} & = & \text{TRUE} \\
\text{Subset(NewSet, } S_1) & = & \text{TRUE} \\
\text{Subset(Insert}(S_1,\, i),\, S_2) & = & \text{if Member}(S_2,\, i) \\
 & & \text{then Subset}(S_1,\, S_2) \\
 & & \text{else FALSE}
\end{array}
$$

Note 1: The Difference operations could also have been implemented using Delete instead of Insert with a few modifications.

Note 2: This implementation assumes that Subset(S_1, S_2) asks is $S_1 \subseteq S_2$.

7. Axiomatic Specification to Change a Set to a Bag

   Common mistakes:

   - Only including the case to count items in your bag when you have just inserted that item.

$$
\begin{array}{lll}
\text{Delete:} & \text{Bag} \times \text{Item} & \to \text{Bag} \\
\text{Count:} & \text{Bag} \times \text{Item} & \to \text{Natural Number}
\end{array}
$$

$$
\begin{array}{lcl}
\text{Delete(NewBag, } i) & = & NewBag \\
\text{Delete(Insert}(B,\, i_1,\, i_2)) & = & \text{if } i_1 = i_2 \\
 & & \text{then } B \\
 & & \text{else Insert(Delete}(B,\, i_2)) \\
\text{Count(NewBag, } i) & = & 0 \\
\text{Count(Insert}(B,\, i_1),\, i_2) & = & \text{if } i_1 = i_2) \\
 & & \text{then Count}(B,\, i_2) + 1 \\
 & & \text{else Count}(B,\, i_2)
\end{array}
$$

8. Axiomatic Specification of SubString Operation

   Common mistakes:

   - Not recursively iterating through the string correctly.

- Passing the Append operation a character and a string when it requires two string operands.
- Off-by-one errors as i and j change as you iterate through the string.

$$\text{SubString:} \quad \text{String} \times \text{Natural} \times \text{Natural} \quad \rightarrow \text{String}$$

Assumptions for the first solution:

- The indicies of the string start at 1.
- i denotes the string index of the starting substring character.
- j denotes the string index of the ending substring character.

$$\text{SubString(NewString, } i, j) \quad = \quad \mathit{NewString}$$

SubString (Append(Add(NewString, c_1), Add(S, c_2)) i, j) =
if $i >$ Length(Append(Add(NewString, c_1), Add(S, c_2)))
then error
else if $j >$ Length(Append(Add(NewString, c_1), Add(S, c_2)))
then error
else if $j < i$
then error
else if $i = 1$ and $j =$ Length(Append(Add(NewString, c_1), Add(S, c_2)))
then Append(Add(NewString, c_1), Add(S, c_2))
else if $i > 1$
then SubString(Add(S_1, c_2, i-1, j-1))
else if $j <$ Length(Append(Add(NewString, c_1), Add(S, c_2)))
then SubString(Append(Add(NewString, c_1),S), i, j)

Assumptions for the second solution:

- The indicies of the string start at 1.
- i denotes the string index of the starting substring character.
- j denotes the number of character to get starting from i.

SubString(NewString, i, j) = *NewString*
SubString(Add(S, c), i, j) = if $j = 0$ or i+j-1 $>$ Length(Add(S, c))
then error
else if i+j-1 = Length(Add(S, c))
then Add(SubString(S, i, j-1), c)
else SubString(S, i, j)