Final Exam -- Sample Solutions

Problem 1.

(a) true -- immediate proof by the Assignment Axiom
(b) false -- for X=2 the pre-condition is true, but this results in X= 2 and Y= -1 after execution so X-Y = 3, not less than 0
(c) false -- for X=0 and Y=1 the pre-condition is true, but this results in X=2 and Y=1 in the post-condition making it false
(d) true -- immediately proven *partially* correct (i.e., when it halts) by the While-rule using {true} as the trivial invariant (but only halts when X=0 initially, and that's the only time the assertion has force)
(e) false — for X=-1 and Y=-1, the pre-condition is true, but then X=-2 and Y=-1 in the post-condition and it is false

Problem 2.

The Z function type assumed here is f:X ⇸ X (i.e., total function). This has the great advantage that there are no exceptions for ApplyToAll (f can be applied to every item of every buffer yielding a value correctly typed for insertion in that buffer). Therefore, the specification consists of a single scheme. ApplyToAll could accommodate substantially more general function types (e.g., partial, differing domain/range, as long as the function can be applied to each item actually appearing in a particular buffer), but then exceptions arise and require treatment so that is not a wise choice for an exam answer.

In Z, the buffer type is **seq** X and that in turn is a function from natural numbers to X and therefore is viewed as a set of pairs. Since the argument is a function, it can be viewed as another set of pairs. So the exit values of the buffer can also be expressed as a set of pairs using set notation in Z.

ApplyToAll[X]

Δ Buffer
f? : X ⇸ X

buffer' = { k ↦ y | k:1..size ∧ k ↦ x ∈ buffer ∧ x ↦ y ∈ f?}
max_size' = max_size

The first post-condition assures both that he size of the exit value of the buffer is correct (same as entry size), and that the elements in the exit value of the buffer are related to the entry values by applying the function. The second post-condition requires that max_size remain unchanged. Since no pre-conditions are needed, there can be no exceptions.

Problem 3.

In Z, the BufferIn operation produces a pair of results (updated Buffer with added item at end, and Report), and the BufferOut operation produces three results (updated Buffer, first Item, and Report). This could be modeled with functions that produce a pair and a

triple, but then hidden functions (plus equations describing them) would be needed to access components of these compound results. Instead we use multiple functions to correspond to each of these operations, each function producing one of the components. The different functions for each operation are signified by a subscript -- B (for Buffer component), X (for item component), or R (for Report component). For example, $Bufferin_B(b,x)$ produces the buffer component of the Z BufferIn operation, and $BufferIn_R(b,x)$ produces the report component of the same operation.

Finally, in Z the max_size attribute was understood to vary between buffers, but it was left implicit how this was determined. This is made explicit in this ADT version by turning the Z constant BufferInit into a function whose argument is the max_size of a created buffer result. Then the size and max_size attributes are both functions in the ADT.

The sorts are Buffer (the TOI), $\mathbb{N}$ (pre-defined as in Z), $\mathbb{N}_1$ (pre-defined as in Z), X (parameter), and Report (since conditional equations are used, Boolean is effectively a hidden sort).

Signature (all visible)
BufferInit: $\mathbb{N}_1 \rightarrow$ Buffer
$BufferIn_B$: Buffer $\times$ X $\rightarrow$ Buffer
$BufferIn_R$: Buffer $\times$ X $\rightarrow$ Report
$BufferOut_B$: Buffer $\rightarrow$ Buffer
$BufferOut_X$: Buffer $\rightarrow$ X
$BufferOut_R$: Buffer $\rightarrow$ Report
max_size: Buffer $\rightarrow \mathbb{N}_1$
size: Buffer $\rightarrow \mathbb{N}$
ok: $\rightarrow$ Report
full: $\rightarrow$ Report
empty: $\rightarrow$ Report

This specification is based on taking BufferInit and BufferIn as constructors -- every buffer value can be formed by these two operations, so definitions are completed by considering arguments consisting of these two cases. As a general guide, note that Buffers and Queues are essentially the same -- both are FIFO mechanisms. Hence, the equations needed for the Buffer function descriptions have the same basic organization as those appearing in the Queue ADT.

Equations
max_size(BufferInit(n)) = n
$max\_size(BufferIn_B(b,x)) = max\_size(b)$
size(BufferInit(n)) = 0
$size(BufferIn_B(b,x))$ = **if** size(b)<max_size(b)
then 1+size(b) **else** size(b)
$BufferIn_B(b,x)$ = **if** size(b)<max_size(b)

then $BufferIn_B(b,x)$ **else** b

$BufferIn_R(b,x))$ = **if** size(b)<max_size(b)
then ok **else** full

$BufferOut_B(BufferInit(n))$ = $BufferInit(n)$

$BufferOut_B(BufferIn_B(b,x)))$ = **if** size(b)=0 **then** b
else $BufferIn_B(BufferOut_B(b),x))$

$BufferOut_X(BufferIn_B(b,x)))$ = **if** size(b)=0 **then** x **else** $BufferOut_B(b)$

$BufferOut_R(b)$ = **if** size(b)=0 **then** empty **else** ok

In Z there are no "error" buffer values since the Z specification indicates the buffer is unchanged in exceptional cases. All the exceptions in this specification are identified through the size function. Also, in the case of an empty buffer in BufferOut, the Z specification leaves the resulting item value (x!) unspecified. The same is done here leaving this specification not sufficiently complete -- e.g., no equation evaluates $BufferOut_X(BufferInit(n))$.

Problem 4.

(a) The rewriting system is terminating but not confluent. Each of the rewriting rules removes one or more occurrences of 's'. 'p', or 'even' and so terminates in a number of steps no greater than the number of these operations in the term.

Equivalent ground terms (i.e., no variables) of sort N do have a unique normal form. To begin, note that these equivalence classes are:

- $[z] = \{z, s(p(z)), p(s(z)), s(s(p(p(z)))), ... \}$ -- all terms with an equal number of applications of 's' and 'p' in any order, by using the first two equations
- $[s^k(z)] = \{s^k(z), p(s^{k+1}(z)), ... \}$ -- all terms with $k \geq 1$ more 's' than 'p' in any order, by using the first two equations
- $[p^k(z)] = \{p^k(z), s(p^{k+1}(z)), ...\}$ -- similarly, all terms with $k \geq 1$ more 'p' than 's' in any order, by using the first two equations

All the terms in one of these classes have the first term written as their normal form.

However, not all equivalent ground terms of sort Boolean have a single normal form. For instance,

even(p(z))
= even(s(s(p(z))) by the last equation
= even(s(z)) by the first equation.

But even(p(z)) is its own normal form (no rewriting possible), while the normal form of even(s(z)) is false. Therefore this system is not confluent and hence not canonical

(b) Since sufficient completeness is an ADT property (not a property of the rewriting technique for animation), for this part we must switch from considering rewrite rules to considering the corresponding equations.

The ADT is sufficiently complete. The Boolean ground terms are of the form even(T), where T is a ground term of sort N. We have already noted the equivalence classes of sort N in part(a). If T is a term in class [z], then as previously noted, its normal

form is z and so even(T) = even(z) = true. If T is in one of the classes [$s^k(z)$], then its normal form is $s^k(z)$ and so by repeatedly applying equation five, the number of occurrences of 's' is reduced to either 0 or 1 and then true or false (respectively) occurs. Finally, if T is a term in one of the classes [$p^k(z)$], its normal form is $p^k(z)$, and even($p^k(z)$) = even(s(s($p^k(z)$))) = even($p^{k-2}(z)$] = ... until reaching either even(z) = true, or even(p(z)) = even(s(s(p(z)))) = even(s(z)) = false. Hence in every case, the term is equivalent to a Boolean value and the ADT is sufficiently complete.

Problem 5.
The table below shows the configuration, occurring events, variable value, generated events, and action at each step. Recall that changes to variables caused by a state's actions or events do not take effect during that step but only in the next step. Also, note that "transitions" are changes of **state**, and connectors are **not** states but indicate branching determined by various conditions associated with the same event -- since AB is an and-chart, configurations that appear in the trace are pairs of the basic states A1, A2, B1, and B2.

step	config	ev occuring	val(N)	gen ev	action
0	<A1,B2>	get(N)	0		put(0)
1	<A1,B2>	get(N)	2		put(2)
2	<A1,B2>	get(N)	1	GOB	
3	<A2,B2>	GOB	4		put(8)
4	<A2,B1>	get(N)	4	GOA	
5	<A2,B2>	GOA	3		put(3)
6	<A1,B2>		3		

Hence the output produced from input <2,1,4,3> is <0,2,8,3>