Wes Rupert – 2/11/13
EECS 345 – Hw 02

1) <S> → <V> = <Q> | <Q>
   <V> → x | y | z
   <Q> → <O> ? <Q> : <Q> | <O>
   <O> → <O> || <A> | <A>
   <A> → <A> && <N> | <N>
   <N> → ! <P> | <P>
   <P> → (<S>) | <B>
   <B> → true | false
2)
   a. Yes. See Fig. 1
   b. No. You cannot have two ‘y’s in a row, as <A> y → x <A> y | x y, which only has <A> y and x y substrings.
   c. No. Since the ‘y’s must be paired with the ‘x’s to become ‘<S>’s, there is no way to group two consecutive ‘<S>’s.
   d. No. Since the ‘y’s must be paired with the ‘x’s to become ‘<S>’s, and the second ‘w’ must be paired with the ‘xyz’ (<S> → w <S> <B> and <B> → z), there is no way to group two consecutive ‘<S>’s.
   e. Yes. See Fig. 2
3) $M_{state}(\{assign\}, S) = Add\big(M_{name}(\{var\}), M_{value}(\{expr\}, S), M_{state}(\{expr\}, S)\big)$

$$M_{state}(\{while\}, S) = \left\{ \begin{matrix} if\ M_{bool}(\{cond\}, S) = false\ then\ M_{state}(\{cond\}, S) \\ else\ M_{state}\big(while\ \{cond\}\ \{loopbody\}, M_{state}(\{cond\}, S)\big), \\ M_{state}\big(\{loopbody\}, M_{state}(\{cond\}, S)\big) \end{matrix} \right\}$$

4)
   a.
```
// Precondition: x < 2 * y
y = y - x                // x > -2 * (y - x) → x > 2 * x - 2 * y
x = x + 2 * y            // x + 2 * y > 0 → x > -2 * y
// Postcondition: x > 0
```
   b.
```
// Precondition: x ≥ 0 && a * a * x ≥ 2 * b
if (a < 0) then          // Since a < 0, ⇒ a * a * x ≥ 2 * b
   y = a * a * x - b     // a * a * x - b ≥ b → a * a * x ≥ 2 * b
else                     // Since a ≥ 0, ⇒ x ≥ 0
   y = a * x + b         // a * x + b ≥ b → a * x ≥ 0
// Postcondition: y ≥ b
```
   c.
```
// Precondition: A[p] < A[i] < A[q]
if A[i] < low then       // Since A[i] < low, ⇒ low ≤ high < A[q]
   p = p + 1             // A[i] < low ≤ high < A[q]
   t = A[i]              // A[i] < low ≤ high < A[q]
   A[i] = A[p]           // t < low ≤ high < A[q]
   A[p] = t              // t < low ≤ high < A[q]
   i = i + 1             // A[p] < low ≤ high < A[q]
else if A[i] > high then // Since A[i] > high, ⇒ A[p] < low ≤ high
```

```
            q = q - 1                // A[p] < low ≤ high < A[i]
            t = A[i]                 // A[p] < low ≤ high < A[i]
            A[i] = A[q]              // A[p] < low ≤ high < t
            A[q] = t                 // A[p] < low ≤ high < t
          else                       // Since A[i] > low && A[i] < high, ⇒ A[p] < A[i-1] < A[q]
            i = i + 1                // A[p] < low ≤ high < A[q]
          // Postcondition: A[p] < low ≤ high < A[q]
```

5)
```
// Precondition: n ≥ 0 and A contains n elements indexed from 0
bound = n - 1;
while (bound > 0) {
  // Let l = {A[bound] ≤ A[bound + 1] ≤ ... ≤ A[n - 1]}
  t = 0;
  for (i = 0; i < bound-1; i++) { // Precondition:
    // Precondition: none
    if (A[i] > A[i+1]) { // Precondition: A[i] ≥ A[i+1]
      swap = A[i]; // A[i+1] ≤ A[i]
      A[i] = A[i+1]; // A[i+1] ≤ swap
      A[i+1] = swap; // A[i] ≤ swap
      t = i+1; // We swapped, so t = i+1, index of larger A[x]
    }
    // Postcondition: A[i] ≤ A[i+1]
  }
  // Postcondition: A[i] ≤ A[i+1] ≤ ... ≤ A[bound]
  bound = t; // t is highest index of swap
}
// Postcondition: A[0] ≤ A[1] ≤ ... ≤ A[n-1]
```

Let A[x:y] be the array {A[x], A[x+1], ..., A[y-1], A[y]}

Inner loop - let l={A[bound] = max(A[0:bound])}
Case 0: bound = 0
  A[0] = max(A[0:0])
Case k+1: Assume A[k] = max(A[0:k])
  If A[k] ≤ A[k+1], the if statement doesn't execute, and A[k+1] = max(A[0:k+1])
  If A[k] > A[k+1], the if statement swaps them, and A[k+1] = max(A[0:k+1])

Outer loop - Let l={A[bound]≤A[bound+1]≤...≤A[n-1]}
Case 0: bound = n-1
  A[n-1]≤A[n-1]
Case t-1: Assume {A[t]≤A[t+1]≤...≤A[n-1]}
  Since inner loop moves the maximum value of A[0:t-1] to the t-1 index, A[0:t-1] ≤ A[t:n-1]
  Since the for loop makes A[t-1] = max(A[0:t-1]), {A[t-1] ≤ A[t] ≤ ... ≤ A[n-1]}

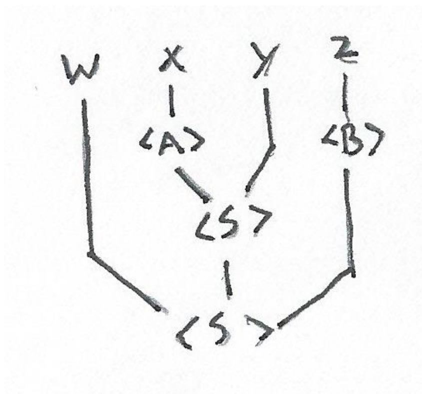

Fig. 1

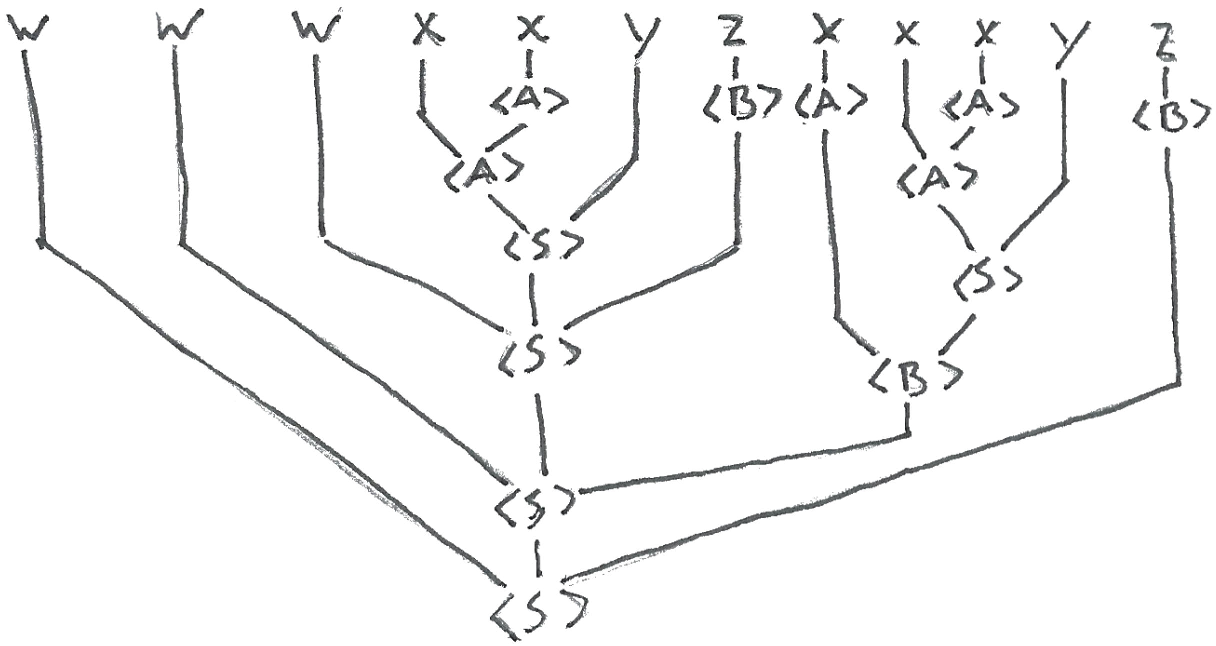

Fig. 2