Rebecca Frederick
EECS 345
Written Exercise 1
February 4, 2016

1. <C> $\rightarrow$ <V> = <D> | <V>

   <V> $\rightarrow$ x | y | z

   <D> $\rightarrow$ <E> ? <D> : <D> | <E>

   <E> $\rightarrow$ <E> || <F> | <F>

   <F> $\rightarrow$ <F> && <G> | <G>

   <G> $\rightarrow$!<G> | <H>

   <H> $\rightarrow$ (<H>) | <I>

   <I> $\rightarrow$ true | false

2. Static Semantic Attributes:

type	=	{integer, double}	(synthesized)
typetable(<var>)	=	{integer, double, error}	(inherited)
inittable(<var>)	=	{true, false, error}	(inherited)
typebinding	=	(<var>, {integer, double})	(synthesized)
initialized	=	(<var>, {true, false})	(synthesized)

   Attribute Rules:

<start$_1$> $\rightarrow$ <stmt$_3$> ; <start$_3$>
<start$_1$>.type := N/A
<start$_1$>.typetable(<var>) := <stmt$_3$>.typetable
<start$_1$>.inittable(<var>) := <stmt$_3$>.initvar
<start$_1$>.typebinding := N/A
<start$_1$>.initialized := N/A

<stmt$_3$>.type := N/A
<stmt$_3$>.typetable := <start$_1$>.typetable
<stmt$_3$>.inittable := <start$_1$>.inittable
<stmt$_3$>.typebinding := N/A
<stmt$_3$>.initialized := N/A
<start$_3$>.type := N/A
<start$_3$>.typetable := <stmt$_3$>.typetable $\cup$ <start$_1$>.typetable
<start$_3$>.inittable := <stmt$_3$>.inittable $\cup$ <start$_1$>.inittable
<start$_3$>.typebinding := N/A
<start$_3$>.initialized := N/A
<start$_2$> $\rightarrow$ <stmt$_4$>
<start$_2$>.type := N/A
<start$_2$>.typetable(<var>) := $\emptyset$
<start$_2$>.inittable(<var>) := $\emptyset$
<start$_2$>.typebinding := N/A
<start$_2$>.initialized := N/A

<stmt$_4$>.type := N/A
<stmt$_4$>.typetable := <start$_2$>.typetable
<stmt$_4$>.inittable := <start$_2$>.inittable

<stmt$_4$>.typebinding := N/A
<stmt$_4$>.initialized := N/A
<start$_3$>.type := N/A
<stmt$_1$> $\rightarrow$ *<declare$_2$>*
<stmt$_1$>.type := N/A
<stmt$_1$>.typetable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_1$>.inittable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_1$>.typebinding := N/A
<stmt$_1$>.initialized := N/A

<stmt$_2$> $\rightarrow$ *<assign$_2$>*
<stmt$_2$>.type := N/A
<stmt$_2$>.typetable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_2$>.inittable(<var>) := $\emptyset$(inherited from <start>)
<stmt$_2$>.typebinding := N/A
<stmt$_2$>.initialized := N/A

<declare$_1$>.type :=
<declare$_1$>.typetable(<var>) :=
<declare$_1$>.inittable(<var>) :=
<declare$_1$>.typebinding :=
<declare$_1$>.initialized :=

<type$_1$>.type :=
<type$_1$>.typetable(<var>) :=
<type$_1$>.inittable(<var>) :=
<type$_1$>.typebinding :=
<type$_1$>.initialized :=

<type$_2$>.type :=
<type$_2$>.typetable(<var>) :=
<type$_2$>.inittable(<var>) :=
<type$_2$>.typebinding :=
<type$_2$>.initialized :=

<assign$_1$>.type :=
<assign$_1$>.typetable(<var>) :=
<assign$_1$>.inittable(<var>) :=
<assign$_1$>.typebinding :=
<assign$_1$>.initialized :=

<expression$_1$>.type :=
<expression$_1$>.typetable(<var>) :=
<expression$_1$>.inittable(<var>) :=
<expression$_1$>.typebinding :=
<expression$_1$>.initialized :=

<expression$_2$>.type :=
<expression$_2$>.typetable(<var>) :=
<expression$_2$>.inittable(<var>) :=
<expression$_2$>.typebinding :=
<expression$_2$>.initialized :=

<expression$_3$>.type :=

<expression$_3$>.typetable(<var>) :=
<expression$_3$>.inittable(<var>) :=
<expression$_3$>.typebinding :=
<expression$_3$>.initialized :=

<expression$_4$>.type :=
<expression$_4$>.typetable(<var>) :=
<expression$_4$>.inittable(<var>) :=
<expression$_4$>.typebinding :=
<expression$_4$>.initialized :=

<expression$_5$>.type :=
<expression$_5$>.typetable(<var>) :=
<expression$_5$>.inittable(<var>) :=
<expression$_5$>.typebinding :=
<expression$_5$>.initialized :=

<value$_1$>.type :=
<value$_1$>.typetable(<var>) :=
<value$_1$>.inittable(<var>) :=
<value$_1$>.typebinding :=
<value$_1$>.initialized :=

<value$_2$>.type :=
<value$_2$>.typetable(<var>) :=
<value$_2$>.inittable(<var>) :=
<value$_2$>.typebinding :=
<value$_2$>.initialized :=

<value$_3$>.type :=
<value$_3$>.typetable(<var>) :=
<value$_3$>.inittable(<var>) :=
<value$_3$>.typebinding :=
<value$_3$>.initialized :=
<value$_4$>.type :=
<value$_4$>.typetable(<var>) :=
<value$_4$>.inittable(<var>) :=
<value$_4$>.typebinding :=
<value$_4$>.initialized :=

Table 1: Attribute Rules

3. (a) 'The type of the expression must match the type of the variable in all assignment statements'

    1. <assign$_1$>: <var>.type = <expression$_3$>.type

   (b) 'A variable must be declared before it is used'

    1. <assign$_1$>: <var>.typetable != 'Error'

   (c) 'A variable must be assigned a value as its first use in the program'

    1. <assign$_1$>: if <var>.inittable = 'Error'

4. Loop Invariants:

Outer (while) Loop Goal: The elements $A[0 \ldots n-1]$ are sorted in non-decreasing order

Outer (while) Loop Invariant: The elements $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$ the elements $A[t \ldots bound-1]$ have yet to be sorted.

(The last condition may be redundant but I felt it necessary to include t in the outer loop invariant since it is initiallized outside of the inner loop and also interacts with a variable (*bound*) in the outer loop.)

Inner (for) Loop Goal: the elements $A[t \ldots n-1]$ are sorted in non-decreasing order

Inner (for) Loop Invariant: The elements $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$
$A[0 \ldots i] \leq A[t] \wedge$
$A[t] \leq A[bound]$.

Precondition: $n \geq 0$ and A contains n elements indexed from 0

```
bound = n;
while (bound > 0) {

  // Assume Outer Loop Invariant is true
  t = 0;

  for (i = 0; i < bound - 1; i++) {

    // Assume Inner Loop Invariant is true
    if (A[i] > A[i+1]) {

      // WP (Inner):
      // $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$
      // $A[0 \ldots i-1] \leq A[i]$ $\wedge$
      // $A[i] \leq A[bound]$
      swap = A[i];

      // WP (Inner):
      // $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$
      // $A[0 \ldots i-1] \leq A[i+1]$ $\wedge$
      // $A[i+1] \leq A[bound]$
      A[i] = A[i+1];

      // WP (Inner):
      // $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$
      // $A[0 \ldots i] \leq$ swap $\wedge$
      // swap $\leq A[bound]$
      A[i+1] = swap;

      // WP (Inner):
      // $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$
      // $A[0 \ldots i] \leq A[i+1]$ $\wedge$
      // $A[i+1] \leq A[bound]$
      t = i + 1;
    }
    // (loop termination: i=bound-1, t='the last i+1 for which $A[i] > A[i+1]$')
    // i=bound-1 $\wedge$ $A[t] \leq A[bound]$ $\wedge$ $A[0 \ldots i] \leq A[t]$ $\wedge$
    //     $A[bound \ldots n-1]$ are in non-decreasing order $\rightarrow$
    //         $A[t \ldots n-1]$ are sorted in non-decreasing order
    // i++
  }
  // WP (Outer):
  // $A[bound \ldots n-1]$ are in non-decreasing order $\wedge$
  //     $A[t \ldots bound-1]$ have yet to be sorted
  bound = t;
```

```
}
// (loop termination: bound=0, t=0)
// bound=0  ∧
// A[0...n−1] are sorted in non-decreasing order  ∧
//    A[0···−1] have yet to be sorted (trivially true)  →
//         array A is sorted in non-decreasing order
```

Postcondition: $A[0] \leq A[1] \leq \cdots \leq A[n-1]$ (i.e., array A is sorted in non-decreasing order)

5. M_{state}(<var> = <expression>, S) =

```
{
    // test that <var> is a legal name in the language
    if M_name(<var>) = 'Error'
        return 'Error'

    // test that <var> has already been declared
    if Lookup(M_name(<var>), S) = 'Error'
        return 'Error'

    // calculate the value of <expression> using the old state
    V = M_value(<expression>, S)
    if V = 'Error'
        return 'Error'

    // calculate a new state including any side effects from evaluating <expression>
    S_1 = M_state(<expression>, S)

    // remove <var> from the new state
    Remove(M_name(<var>), S)

    // return the new state with the updated value of <var> added
    return Add(M_name(<var>), V, S_1)

}
```

M_{state}(if <condition> then <statement$_1$> else <statement$_2$>, S) =

```
{
    S_1 = M_state(<condition>, S)

    if M_boolean(<condition>, S_1) = true
        return M_state(<statement_1>, S_1)
    else if M_boolean(<condition>, S_1) = false
        return M_state(<statement_2>, S_1)
    else
        return 'Error'
}
```

M_{state}(while <condition> <loop body>, S) =

```
{
    S_1 = M_state(<condition>, S)

    if M_boolean(<condition>, S_1) = true
        // evaluate the loop body and call the while-loop again
        return M_state(while <condition> <loop body>, M_state(<loop body>, S_1))
    else if M_boolean(<condition>, S_1) = false
        return S_1
    else
        return 'Error'
```

```
}
```