SWEN90010: HIGH INTEGRITY SYSTEMS ENGINEERING
DEPARTMENT OF COMPUTING AND INFORMATION SYSTEMS
THE UNIVERSITY OF MELBOURNE

Workshop 9 sample solutions

Exercises

3. The proof is failing because the line:

```
1  Result := Input * 10;
```

   can produce an integer overflow. That is, if `Input * 10` is greater than `Integer'Last`, then the computation will give an unexpected value.

4. If we uncomment the commented lines, the constraint that cannot be proved refers not to the line above, but to the conditions in the branch:

```
1     if Input * 10 <= Integer'Last and
2        Input * 10 >= Integer'First then
```

   Again, the problem is that `Input * 10` could overflow. Although this is not assigned to a variable, our underlying architecture still only handles integers of a certain size. To check that the computation will not overflow, we instead have to switch around the computation to use division instead of multiplication:

```
1   procedure Task3Procedure(Input : in Integer; Result : out Integer) is
2   begin
3      if Input <= Integer'Last / 10 and
4        Input >= Integer'First / 10 then
5         Result := Input * 10;
6      else
7         Result := Input;
8      end if;
9   end Task3Procedure;
```

5. For `task4.adb`, the problem again relates to an overflow: in this case, that the expression `AnArray(AnIndex) + 1` will overflow if `AnArray(AnIndex)` is equal to `Integer'Last`.

   To correct this, we insert a new guard:

```
1   procedure Task4Procedure(AnArray : in out MyArray; AnIndex : in Index) is
2   begin
3      if AnArray(AnIndex) < Integer'Last then
4         AnArray(AnIndex) := AnArray(AnIndex + 1);
5      end if;
6   end Task4Procedure;
```

   What this means is that, using the SPARK tools, if all of the conjectures are successfully proved, then we have *no integer overflows in our program* — a very useful (and sometimes imperative!) property.

   In the next workshop, we'll look at a different solution that involves using preconditions.

8. For the expression `AnArray(AnIndex + 1)`, the problem is different. This error here relates to that fact that the array could be indexed out of bounds. That is, if `AnIndex` is equal to `Index'Last`, then the expression `AnArray(AnIndex + 1)` will index the array out of bounds. Thus, the SPARK toolset could

not prove that an out-of-bounds index error was not possible. We can use much the same solution as above to eliminate this:

```
1     if AnIndex < Index'Last then
```

What this means is that, using the SPARK tools, if all of the conjectures are successfully proved, then we have *no index out of bounds errors in our program*. A very powerful property – and one that is difficult to detect with tests alone.