Instructor: Matt Fredrikson TA: Ryan Wagner

Incremental Proof Development in Dafny

These notes continue the discussion from Lecture 17 on proving the correctness of imperative code in Dafny. Before we focused on partial correctness, and didn't worry about termination. Here, we'll focus on termination. We'll start with the termination argument for `BubbleSort`, for which we proved partial correctness before. We'll then take a look at a different method that naturally requires a different kind of well-founded relation to prove termination.

Proving Termination

After proving that the method satisfies its postcondition whenver it terminates, in order to prove total correctness we must prove that it always terminates when the preconditions are satisfied. Notice that we did not annotate our program with any ranking functions. Dafny was able to figure out the appropriate ranking functions for our code, and insert them automatically. In this case, the following ranking functions would work well.

```
method BubbleSort(a: array<int>)
  modifies a
  requires a != null
{
  var i := a.Length - 1;
  while(i > 0)
    invariant 0 <= i+1
    decreases i+1, i+1
  {
    var j := 0;
    while(j < i)
      invariant 0 <= i+1 && 0 <= i - j
      decreases i+1, i-j
    {
      if(a[j] > a[j+1])
      {
        a[j], a[j+1] := a[j+1], a[j];
      }
      j := j + 1;
    }
    i := i - 1;
  }
}
```

However, from the perspective of a programmer writing code in Dafny, one could also write:

```
method BubbleSort(a: array<int>)
  modifies a
  requires a != null
{
  var i := a.Length - 1;
  while(i > 0)
    invariant -1 <= i
    decreases i
  {
    var j := 0;
    while(j < i)
      invariant 0 <= i - j
      decreases i-j
    {
      ...
      j := j + 1;
    }
    i := i - 1;
  }
}
```

We could drop the invariants as well, as Dafny is able to infer them without assistance. However, would these ranking functions work if we tried to prove them by hand? Consider the basic path leaving the inner loop, and going back to execute the outer loop again.

```
invariant 0 <= i - j
decreases i - j
assume j >= i;
i := i - 1;
decreases i
```

The corresponding verification condition is

$$\begin{aligned} & 0 \le \mathtt{i} - \mathtt{j} \to \mathsf{wlp}(\mathtt{assume}\ \mathtt{j} > \mathtt{i}; \mathtt{i} := \mathtt{i} - 1, \mathtt{i} < \mathtt{i}' - \mathtt{j}')[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\ \Leftrightarrow\ & 0 \le \mathtt{i} - \mathtt{j} \land \mathtt{j} \ge \mathtt{i} \to \mathtt{i} - 1 < \mathtt{i} - \mathtt{j} \end{aligned}$$

This is not valid, and in fact the consequent is only true when $\mathtt{i} = \mathtt{j} = 0$.

The simpler ranking functions are accepted by Dafny because it does not generate exactly the same verification conditions that we do when proving termination by hand. To better understand how Dafny constructs a termination argument, consider the following program.

```
method BubbleSort(a: array<int>)
  modifies a
  requires a != null
{
  var i := a.Length - 1;
  while(i > 0)
    invariant -1 <= i
    decreases i
  {
    ghost var rankval_0 := i;
    var j := 0;
    while(j < i)
      invariant 0 <= i - j
      decreases i-j
    {
      ghost var rankval_1 := i - j;
      ...
      j := j + 1;
      assert i - j < rankval_1;
    }
    i := i - 1;
    assert i < rankval_0;
  }
}
```

Roughly, this corresponds to the checks performed by Dafny given these `ranking` annotations. We could prove this by hand as well, showing that the assertions always hold by generating additional verification conditions for the assertions. We would arrive at the same result as we do using the "standard" termination approach with the first set of ranking functions, but doing so would require developing additional invariants (e.g., that `i` is not modified by the inner loop), and would not necessarily lead to a simpler proof in the end.

When developing termination proofs by hand, you should use the uniform approach we've discussed in class rather than the method involving ghost state used by Dafny. When working with Dafny, you may find it helpful to realize that this is how it constructs its termination argument to gain insight into why your annotations are not being accepted by the verifier. However, if you construct a valid ranking function that you are able to prove using the standard method, then you should in most cases be able to use it as an annotation in your code that will be accepted by the verifier. If the verifier does not accept what appears to be a perfectly good ranking function, then you should proceed to derive the relevant verification conditions in this manner to check your work. Usually, you will be able to identify a path on which your function does not decrease, or in certain cases, a verification condition that looks as though it will be difficult for the decision procedure to automatically discharge (e.g., involving complex terms from the theory of integers or reals).

For completeness, let's walk through the termination argument on the properly-annotated method (i.e., the first version of `BubbleSort` listed in this section). Notice that we do not have a ranking function on the method itself, as it is not recursive. Our termination argument will focus solely on the loops contained in the body, so we can ignore the basic paths that enter and exit the outer loop. We begin with the basic path starting at the top of the outer loop, entering the inner loop.

```
invariant 0 <= i+1
decreases i+1, i+1
assume i > 0;
j := 0;
decreases i+1, i-j
```

Our verification condition is given by:

$$\begin{aligned}
&0 \le \mathtt{i}+1 \to \mathsf{wlp}(\mathtt{assume\ i} > 0; \mathtt{j} := 0, (\mathtt{i}+1, \mathtt{i}-\mathtt{j}) < (\mathtt{i}'+1, \mathtt{i}'+1))[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\
\Leftrightarrow\ &0 \le \mathtt{i}+1 \to \mathsf{wlp}(\mathtt{assume\ i} > 0, (\mathtt{i}+1, \mathtt{i}-0) < (\mathtt{i}'+1, \mathtt{i}'+1))[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\
\Leftrightarrow\ &0 \le \mathtt{i}+1 \to (\mathtt{i} > 0 \to (\mathtt{i}+1, \mathtt{i}) < (\mathtt{i}'+1, \mathtt{i}'+1))[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\
\Leftrightarrow\ &0 \le \mathtt{i}+1 \to (\mathtt{i}+1, \mathtt{i}) < (\mathtt{i}+1, \mathtt{i}+1)
\end{aligned}$$

The last step removes the additional antecedent $\mathtt{i} > 0$, because it is implied by $0 \le \mathtt{i}+1$, as well as substituting the primed variables. We see that it is valid because of the lexicographic ordering, as $\mathtt{i}+1 = \mathtt{i}+1$ and $\mathtt{i} < \mathtt{i}+1$.

The next basic path corresponds to a non-exiting iteration of the inner loop. The comments label the commands, so that we do not need to write them in full when constructing the verification condition.

```
invariant 0 <= i+1 && 0 <= i - j
decreases i+1, i-j
assume j < i;                    // c0
assume a[j] > a[j+1];            // c1
a[j], a[j+1] := a[j+1], a[j];    // c2
j := j + 1;                      // c3
decreases i+1, i-j
```

The verification condition is:

$$\begin{aligned}
&0 \le \mathtt{i}+1 \land 0 \le \mathtt{i} - j \to \mathsf{wlp}(c_0; c_1; c_2; c_3, (\mathtt{i}+1, \mathtt{i}-\mathtt{j}) < (\mathtt{i}'+1, \mathtt{i}'-\mathtt{j}'))[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\
\Leftrightarrow\ &0 < \mathtt{i} \land 0 \le \mathtt{i} - j \land \mathtt{j} < \mathtt{i} \land \mathtt{a}[\mathtt{j}] > \mathtt{a}[\mathtt{j}+1] \to (\mathtt{i}+1, \mathtt{i}-(\mathtt{j}+1)) < (\mathtt{i}+1, \mathtt{i}-\mathtt{j})
\end{aligned}$$

This is valid, because $\mathtt{i} - (\mathtt{j}+1) < \mathtt{i} - \mathtt{j}$. Notice that the statements involving the array have no bearing on the validity of this condition. In the interest of keeping the verification condition simpler, we could have just written:

$$0 < \mathtt{i} \land 0 \le \mathtt{i} - j \land \mathtt{j} < \mathtt{i} \to (\mathtt{i}+1, \mathtt{i}-(\mathtt{j}+1)) < (\mathtt{i}+1, \mathtt{i}-\mathtt{j})$$

However, when doing these proofs by hand, it's best to write the entire VC out at first, and simplify afterwards with an additional step that may help to clarify the reasoning. Moving onto the next basic path:

```
invariant 0 <= i+1 && 0 <= i - j
decreases i+1, i-j
assume j < i;           // c0
assume a[j] <= a[j+1];  // c1
j := j + 1;             // c2
decreases i+1, i-j
```

The verification condition is identical to the previous one with the exception of the condition in the `assume` statement, which is irrelevant to the termination argument:

$$\begin{aligned} & 0 \le \mathtt{i} + 1 \land 0 \le \mathtt{i} - j \to \mathsf{wlp}(c_0; c_1; c_2, (\mathtt{i}+1, \mathtt{i}-\mathtt{j}) < (\mathtt{i}'+1, \mathtt{i}'-\mathtt{j}'))[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\ \Leftrightarrow \quad & 0 < \mathtt{i} \land 0 \le \mathtt{i} - j \land \mathtt{j} < \mathtt{i} \land \mathsf{a}[\mathtt{j}] \ge \mathsf{a}[\mathtt{j}+1] \to (\mathtt{i}+1, \mathtt{i}-(\mathtt{j}+1)) < (\mathtt{i}+1, \mathtt{i}-\mathtt{j}) \end{aligned}$$

The final basic path that we need to consider is:

```
invariant 0 <= i+1 && 0 <= i - j
decreases i+1, i-j
assume j >= i;              // c0
i := i - 1;                 // c1
decreases i+1, i+1
```

This yields the verification condition:

$$\begin{aligned} & 0 \le \mathtt{i} + 1 \land 0 \le \mathtt{i} - j \to \mathsf{wlp}(c_0; c_1, (\mathtt{i}+1, \mathtt{i}-\mathtt{j}) < (\mathtt{i}'+1, \mathtt{i}'+1))[\mathtt{i}/\mathtt{i}', \mathtt{j}/\mathtt{j}'] \\ \Leftrightarrow \quad & 0 < \mathtt{i} \land 0 \le \mathtt{i} - j \land \mathtt{j} \ge \mathtt{i} \to (\mathtt{i}, \mathtt{i}) < (\mathtt{i}+1, \mathtt{i}+1) \end{aligned}$$

This is obviously valid, so we conclude that the loop in `BubbleSort` terminates.

"Non-standard" Orderings Consider the following method, for which we would like to prove termination.

```
method M(x0: int, y0: int, z0: int)
{
  var x, y, z := x0, y0, z0;
  while x > 0 && y > 0 && z > 0
  {
    if y > x {
      y := z;
      x := *;
      z := x - 1;
    } else {
      z := z - 1;
      x := *;
      y := x - 1;
    }
  }
}
```

This method contains a statement form that we haven't seen before, namely non-deterministic assignment:

```
x := *;
```

The semantics of this statement are that `x` is assigned an arbitrary value from its domain; we can make no assumptions about what the value is, and when reasoning symbolically, we must treat it as a fresh constant.

$$\mathsf{wlp}(\mathtt{x} := *, Q) = Q[\mathtt{x}/\mathtt{x}'] \quad \text{where } \mathtt{x}' \text{ is a fresh symbol not appearing in } Q$$

How do we prove that this method terminates. We can begin by observing that `z` definitely decreases in the `else` branch of the conditional statement. It also seems that `z` may decrease in the `if` branch, because we assign to it $\mathtt{x} - 1$. Perhaps we should try using $\downarrow \mathtt{z}$ as our ranking function.

```
method M(x0: int, y0: int, z0: int)
{
  var x, y, z := x0, y0, z0;
  while x > 0 && y > 0 && z > 0
    decreases z
  {
    if y > x {
      y := z;
      x := *;
      z := x - 1;
    } else {
      z := z - 1;
      x := *;
      y := x - 1;
    }
  }
}
```

Let's first do a quick sanity check that this suffices in the branch that we expect it to work in, considering the basic path corresponding to the `else`. We don't have a loop invariant, to the initial assertion in this path is simply *true*.

```
invariant true
decreases z
assume y <= x      // c0
z := z - 1;        // c1
x := *;            // c2
y := x - 1;        // c3
decreases z
```

Looking at the verification condition for this path,

$$\begin{aligned}
&\mathsf{wlp}(c0; c1; c2; c3, \mathsf{z} < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &\mathsf{wlp}(c0; c1; c2, \mathsf{z} < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &\mathsf{wlp}(c0; c1, \mathsf{z} < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &\mathsf{wlp}(c0, \mathsf{z} - 1 < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &(\mathtt{y} \le \mathtt{x} \to \mathtt{z} - 1 < \mathtt{z})
\end{aligned}$$

This is valid, but we also need to make sure that `z` is bounded from below to ensure well-foundedness. We simply need $0 \le$ `z` in the antecedent of this condition, which we can obtain by adding the appropriate loop invariant.

```
invariant 0 <= z
```

Now let's take a look at the other branch.

```
invariant 0 <= z
decreases z
assume y > x       // c0
y := z;            // c1
x := *;            // c2
z := x - 1;        // c3
decreases z
```

This gives us the verification condition:

$$\begin{aligned}
&0 \le \mathsf{z} \to \mathsf{wlp}(c0; c1; c2; c3, \mathsf{z} < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &0 \le \mathsf{z} \to \mathsf{wlp}(c0; c1; c2, \mathtt{x} - 1 < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &0 \le \mathsf{z} \to \mathsf{wlp}(c0; c1, \mathtt{x}'' - 1 < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &0 \le \mathsf{z} \to \mathsf{wlp}(c0, \mathtt{x}'' - 1 < \mathsf{z}')[\mathsf{z}/\mathsf{z}'] \\
\Leftrightarrow\ &(0 \le \mathtt{z} \land \mathtt{y} > \mathtt{x}) \to \mathtt{x}'' - 1 < \mathtt{z}
\end{aligned}$$

This is not valid, as $\mathtt{x}''$ can take a value greater than `z`. We'll need to refine the verification condition to show that it decreases on this path.

Using a bit of intuition, we observe that a lexicographic ranking function might be a good choice, as we have a natural ranking function for the second branch, so perhaps we can find a function that decreases on the first branch and remains constant on the second. So, we're looking for some function f to insert into the lexicographic function:

$$(f(\mathtt{x}, \mathtt{y}, \mathtt{z}), \mathtt{z})$$

We continue by observing that in the second branch, the condition `y` $>$ `x` remains unchanged: it is false whenever this branch is entered, and remains false because `y` is assigned `x` $- 1$ at the end of it. This is a Boolean-valued function, so if we assume an ordering on the Booleans, we may be able to use this in our first component. We're free to choose any ordering on the Booleans to build our lexicographic order, but according to our reasoning so far, it seems that we should use:

$$\mathit{false} < \mathit{true}$$

With this ordering over the Booleans in mind, our ranking function will be ordered lexicographically over pairs of Booleans and integers:

```
decreases y > x, z
```

To make our newly-constructed ordering completely explicit, let's write down a formula that characterizes whether a pair of tuples $(\mathtt{b}_1, \mathtt{n}_1) < (\mathtt{b}_2, \mathtt{n}_2)$.

$$\begin{array}{rl} & (\mathtt{b}_1, \mathtt{n}_1) < (\mathtt{b}_2, \mathtt{n}_2) \\ \textbf{if and only if} & (\neg \mathtt{b}_1 \land \mathtt{b}_2) \lor ((\mathtt{b}_1 \leftrightarrow \mathtt{b2}) \land \mathtt{n}_1 < \mathtt{n}_2) \end{array}$$

Extending this characterization to match the form of our ranking function, we have:

$$\begin{array}{rl} & (\mathtt{y}_1 > \mathtt{x}_1, \mathtt{z}_1) < (\mathtt{y}_2 > \mathtt{x}_2, \mathtt{z}_2) \\ \textbf{if and only if} & (\neg(\mathtt{y}_1 > \mathtt{x}_1) \land \mathtt{y}_2 > \mathtt{x}_2) \lor ((\mathtt{y}_1 > \mathtt{x}_1 \leftrightarrow \mathtt{y}_2 > \mathtt{x}_2) \land \mathtt{z}_1 < \mathtt{z}_2) \\ \Leftrightarrow & (\mathtt{y}_1 \le \mathtt{x}_1 \land \mathtt{y}_2 > \mathtt{x}_2) \lor ((\mathtt{y}_1 > \mathtt{x}_1 \leftrightarrow \mathtt{y}_2 > \mathtt{x}_2) \land \mathtt{z}_1 < \mathtt{z}_2) \end{array}$$

Let's see if it works on the troublesome basic path from before.

```
invariant 0 <= z
decreases y > x, z
assume y > x         // c0
y := z;              // c1
x := *;              // c2
z := x - 1;          // c3
decreases y > x, z
```

We see that this gives us the verification condition:

$$\begin{array}{rl} & 0 \le \mathtt{z} \to \mathsf{wlp}(c0; c1; c2; c3, (\mathtt{y} > \mathtt{x}, \mathtt{z}) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow & 0 \le \mathtt{z} \to \mathsf{wlp}(c0; c1; c2, (\mathtt{y} > \mathtt{x}, \mathtt{x} - 1) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow & 0 \le \mathtt{z} \to \mathsf{wlp}(c0; c1, (\mathtt{y} > \mathtt{x}'', \mathtt{x}'' - 1) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow & 0 \le \mathtt{z} \to \mathsf{wlp}(c0, (\mathtt{z} > \mathtt{x}'', \mathtt{x}'' - 1) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow & (0 \le \mathtt{z} \land \mathtt{y} > \mathtt{x}) \to (\mathtt{z} > \mathtt{x}'', \mathtt{x}'' - 1) < (\mathtt{y} > \mathtt{x}, \mathtt{z}) \end{array}$$

Is this valid? We can use the formula we derived earlier to decide by replacing the consequent of our verification condition with the requirements imposed by our ordering. This amounts to checking the validity of:

$$(0 \le \mathtt{z} \land \mathtt{y} > \mathtt{x}) \to ((\mathtt{z} \le \mathtt{x}'' \land \mathtt{y} > \mathtt{x}) \lor ((\mathtt{z} > \mathtt{x}'' \leftrightarrow \mathtt{y} > \mathtt{x}) \land \mathtt{x}'' - 1 < \mathtt{z}))$$

This formula is fairly complex to reason about due to the multiple instance of disjunction (don't forget about the instance hiding in the biconditional). We know that because of the antecedent condition $\mathtt{y} > \mathtt{x}$, the first component of the second tuple will be *true*. Thus, we can forget about trying to satisfy the first disjunct:

$$(\mathtt{z} \le \mathtt{x}'' \land \mathtt{y} > \mathtt{x})$$

We'll focus on the second disjunct, which according to our lexicographic ordering, forces the first component of the first tuple to either be *false*, i.e.,

$$\mathtt{z} \le \mathtt{x}''$$

Or if it is not, then $\mathtt{x}'' - 1$ must be less than $\mathtt{z}$, i.e.,

$$\mathtt{z} > \mathtt{x}'' \land \mathtt{x}'' - 1 < \mathtt{z}$$

Putting all of this together, we arrive at a simpler formula whose validity implies that of the VC:

$$\begin{aligned} & (\mathtt{z} \le \mathtt{x}'') \lor (\mathtt{z} > \mathtt{x}'' \land \mathtt{x}'' - 1 < \mathtt{z}) \\ \Leftrightarrow\ & \mathtt{z} > \mathtt{x}'' \to (\mathtt{z} > \mathtt{x}'' \land \mathtt{x}'' - 1 < \mathtt{z}) \\ \Leftrightarrow\ & \mathtt{z} > \mathtt{x}'' \to \mathtt{x}'' - 1 < \mathtt{z} \end{aligned}$$

This is valid, so the ranking function decreases on this path. Even though we don't expect the other basic path to pose a problem, let's go back and make sure that it decreases as it should.

```
invariant 0 <= z
decreases y > x, z
assume y <= x      // c0
z := z - 1;        // c1
x := *;            // c2
y := x - 1;        // c3
decreases y > x, z
```

With our new loop invariant and ranking function, we now have the verification condition:

$$\begin{aligned} & 0 \le \mathtt{z}\mathsf{wlp}(c0; c1; c2; c3, (\mathtt{y} > \mathtt{x}, \mathtt{z}) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow\ & 0 \le \mathtt{z} \to \mathsf{wlp}(c0; c1; c2, (\mathtt{x} - 1 > \mathtt{x}, \mathtt{z}) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow\ & 0 \le \mathtt{z} \to \mathsf{wlp}(c0; c1, (\mathtt{x}'' - 1 > \mathtt{x}'', \mathtt{z}) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow\ & 0 \le \mathtt{z} \to \mathsf{wlp}(c0, (\mathtt{x}'' - 1 > \mathtt{x}'', \mathtt{z} - 1) < (\mathtt{y}' > \mathtt{x}', \mathtt{z}'))[\mathtt{x}/\mathtt{x}', \mathtt{y}/\mathtt{y}', \mathtt{z}/\mathtt{z}'] \\ \Leftrightarrow\ & (0 \le \mathtt{z} \land \mathtt{y} \le \mathtt{x}) \to (\mathtt{x}'' - 1 > \mathtt{x}'', \mathtt{z} - 1) < (\mathtt{y} > \mathtt{x}, \mathtt{z}) \end{aligned}$$

This formula is valid. On the left side of the inequality, we have:

$$(\mathtt{x}'' - 1 > \mathtt{x}'', \mathtt{z} - 1) \text{ reduces to } (\mathit{false}, \mathtt{z} - 1)$$

Keeping in mind the antecedent $\mathtt{y} \le \mathtt{x}$, on the right side of the inequality we have:

$$(\mathtt{y} > \mathtt{x}, \mathtt{z}) \text{ reduces to } (\mathit{false}, \mathtt{z})$$

So, the first component of the lexicographic comparison is an equality, because $\mathit{false} = \mathit{false}$. The second component compares $\mathtt{z} - 1 < \mathtt{z}$, which of course is *true*, making the verification condition valid. This completes the proof.