IP Lecture 5: String Searching

Joe Pitt-Francis

—with thanks to Mike Spivey & Gavin Lowe—

Comparing strings

Suppose we want to compare two `Strings`, `sa` and `sb` to see if they are equal. (Scala allows you to do this using the `==` operator; but it's good to know how to define such things for ourselves.)

It's easiest if we start by converting them into arrays of `Chars`:

```
val a = sa.toArray; val b = sb.toArray
```

We then want a function

```
/** Do a and b hold the same characters?
  * Post: returns a[0..a.size) = b[0..b.size) */
def search(a: Array[Char], b: Array[Char]) : Boolean = ...
```

that tests if the arrays are equal.

Comparing arrays of characters

We can start by checking the arrays are the same size:

```
if(a.size != b.size) return false
```

A `return` statement immediately returns from the function call, skipping the following code.

Comparing arrays of characters

If the arrays are the same size, letting `n = a.size`, we want to test if `a[0..n)=b[0..n)`; more precisely, we want to establish the postcondition `equal = (a[0..n)=b[0..n))`. It seems sensible to compare the elements one at a time, which suggests the invariant

$$I \mathrel{\widehat{=}} \texttt{equal} = (\texttt{a}[0..\texttt{k}) = \texttt{b}[0..\texttt{k})) \wedge 0 \leq \texttt{k} \leq \texttt{n}.$$

This leads to the following straightforward code:

```
var k=0; var equal = true; val n = a.size
while(k<n){
  equal = equal && a(k)==b(k)
  k = k+1
}
// equal = (a[0..n)=b[0..n))
equal
```

- Check the conditions for correctness.

An improved version

With the previous code, if `equal` became `false`, it remains `false` subsequently. But the program continued to check the remaining elements of the array, needlessly. It would be better to continue looping only if `equal` is still true. That suggests the following code

```
var k=0; var equal = true; val n = a.size
while(k<n && equal){
  equal = equal && a(k)==b(k)
  k = k+1
}
// does equal = (a[0..n)=b[0..n)) here?
equal
```

The initialisation and loop body are unchanged, so clearly the same invariant holds. But does this give the right answer, i.e., does `equal = (a[0..n)=b[0..n))` at the end?

A more improved version

The guard of the `while` loop ensures that `equal = true` when we enter the loop body. Hence the assignment

```
    equal = equal && a(k)==b(k)
```

can be simplified:

```
var k=0; var equal = true; val n = a.size
while(k<n && equal){
  equal = a(k)==b(k)
  k = k+1
}
equal
```

A yet more improved version

In fact, we can do without the variable `equal`, and check the condition in the guard.

```
var k=0; val n = a.size
while(k<n && a(k)==b(k)) k = k+1
// (a[0..n) = b[0..n)) = (k=n)
k==n
```

The invariant now is

$$\texttt{a}[0..\texttt{k}) = \texttt{b}[0..\texttt{k}) \wedge 0 \leq \texttt{k} \leq \texttt{n}.$$

- Check the conditions for correctness.

A quick and dirty version

The final version jumps out of the procedure as soon as a mis-match is found.

```
var k=0; val n = a.size
while(k<n)
  if(a(k)!=b(k)) return false else k = k+1
true
```

The keyword **return** is necessary here in order to jump out of the procedure.

The invariant is again

$$\texttt{a}[0..\texttt{k}) = \texttt{b}[0..\texttt{k}) \wedge 0 \leq \texttt{k} \leq \texttt{n}.$$

Reasoning about a function written in this way is a little messy, as one needs to also check that the right think is done at the premature **return**.

String searching

Now consider the problem of searching for an occurrence of one string `pat` of size `K` in another string `line` of length `N`.

More precisely, we want to set a Boolean variable `found` to `true` if $\texttt{line}[i..i+\texttt{K}) = \texttt{pat}[0..\texttt{K})$ for some i; if this is the case, then we must have $0 \leq i \leq \texttt{N} - \texttt{K}$.

Thus we can capture the postcondition as

post: returns `found` s.t.

$$\texttt{found} = \big(\texttt{line}[i..i+\texttt{K}) = \texttt{pat}[0..\texttt{K}), \quad \text{for some } i \in [0..\texttt{N} - \texttt{K} + 1)\big)$$

This suggests using a variable `j` to record the values of i that we've tried so far; i.e. we use the invariant

$$\texttt{found} = \big(\texttt{line}[i..i+\texttt{K}) = \texttt{pat}[0..\texttt{K}), \quad \text{for some } i \in [0..\texttt{j})\big)$$
$$\wedge\ 0 \leq \texttt{j} \leq \texttt{N} - \texttt{K} + 1$$

Towards code

Using invariant:

$$I \mathrel{\widehat{=}} \texttt{found} = \big(\texttt{line}[i..i+\texttt{K}) = \texttt{pat}[0..\texttt{K}), \quad \text{for some } i \in [0..\texttt{j})\big)$$
$$\wedge\ 0 \le \texttt{j} \le \texttt{N} - \texttt{K} + 1$$

we get this code structure:

```
var j = 0; var found = false
while(j <= N-K && !found){
  found = line[j..j+K) == pat[0..K) // pseudocode
  j = j+1
}
// I && (j=N-K+1 || found)
// found = ( line[i..i+K) = pat[0..K) for some i in [0..N-K+1) )
```

The last step

The pseudocode

```
found = line[j..j+K) == pat[0..K)
```

is just an instance of our earlier problem of comparing two arrays of characters for equality, so we can adapt that code.

The search function

```
def search(pat: Array[Char], line: Array[Char]) : Boolean = {
    val K = pat.size; val N = line.size
    // Invariant: I: found = (line[i..i+K) = pat[0..K) for
    //                        some i in [0..j)) and 0 <= j <= N-K
    var j = 0; var found = false
    while(j <= N-K && !found){
      // set found if line[j..j+K) = pat[0..K)
      // Invariant: line[j..j+k) = pat[0..k)
      var k = 0
      while(k<K && line(j+k)==pat(k)) k = k+1
      found = (k==K)
      j = j+1
    }
    // I && (j=N-K+1 || found)
    // found = ( line[i..i+K) = pat[0..K) for some i in [0..N-K+1) )
    found
  }
```

grep

The unix utility `grep` (in its simplest form) takes a string `pat` and a file name `file`, and prints every line from `file` that contains `pat`.

We can use the `search` function to implement `grep`:

```
object Grep{
  /** Does pat appear in line? */
  def search(pat: Array[Char], line: Array[Char]) : Boolean = ...

  def main(args: Array[String]) = {
    require(args.size==2)
    val pat = args(0).toArray
    val file = args(1)
    val lines = scala.io.Source.fromFile(file).getLines
    for(line <- lines)
      if(search(pat,line.toArray)) println(line)
  }
}
```

Summary

- Testing two arrays of characters for equality;
- Testing if one array of characters contains another;
- `grep`.
- Different styles of writing loops.
- Next time: Numbers in decimal.