Correctness Statements Proved in Class

Stefan D Bruda

20 March 2026

We discussed a more complex example that includes a loop as well as array assignment:

```
ASSERT(1 <= n <= max)
int j;
j = 1;
A[0] = 1;
while (j < n) {
    A[j] = A[j-1] + power(2,j);
    j++;
}
ASSERT(ForAll (k=1; k<n) A[k] == power(2,k+1) - 1)
```

This is the apex of verification as presented in the course and expected from you in assignments and the final examination.

While proving this correctness statement one of the following properties of quantified assertions is going to prove very useful:

- We can pull out any "end" of a `ForAll` assertion as a separate term in conjunction with the rest of the `ForAll` assertion. More precisely, for any statement like this:

  ```
  ForAll (k=i; k<=j) P(k)
  ```

  we can pull out as a separate term either the "`i`" end or the "`j`" end. That is, the following statements are both equivalent to the original:

  ```
              ForAll (k=i; k<=j-1) P(k)  &&  P(j)
  P(i)  &&  ForAll (k=i+1; k<=j) P(k)
  ```

  For this particular proof we use the first equivalent statement (we pull out the largest end).

- A similar property holds for `Exists` assertions, except that the terms being pulled out are combined with the remaining `Exists` assertion in a disjunction. That is, the statement:

  ```
  Exists (k=i; k<=j) P(k)
  ```

  is equivalent to both of the following statements:

```
                        Exists (k=i; k<=j-1) P(k)  ||  P(j)
              P(i)  ||  Exists (k=i+1; k<=j) P(k)
```

We do not have existentially quantified assertions in this particular example, but all these properties are worth remembering since they are useful in all the circumstances in which an array is traversed using a simple `while` or `for` loop.

This all being said, here is the proof:

```
ASSERT(1 <= n <= max)
ASSERT(1 <= n)
int j;
ASSERT(1 <= n)
ASSERT(1 <= 1 <= n)
// ForAll trivially true on an empty range
ASSERT(ForAll (k=1; k<1) (A | 0 -> 1)[k] == power(2,k+1) - 1  &&  1 <= 1 <= n)
j = 1;
ASSERT(ForAll (k=1; k<j) (A | 0 -> 1)[k] == power(2,k+1) - 1  &&  1 <= j <= n)
A[0] = 1;
ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1  &&  1 <= j <= n)
while (j < n) {
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           &&  1 <= j <= n && j < n)
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           &&  1 <= j+1 <= n && j < n)
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           &&  1 <= j+1 <= n)
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           && power(2,j+1) - 1 == power(2,j+1) - 1
           &&  1 <= j+1 <= n)
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           && 2 * power(2,j) - 1 == power(2,j+1) - 1
           &&  1 <= j+1 <= n)
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           && power(2,j) - 1 + power(2,j) == power(2,j+1) - 1
           &&  1 <= j+1 <= n)
    // A[j-1] == power(2,j-1+1) - 1 == power(2,j) - 1 from the range
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1
           && A[j-1] + power(2,j) == power(2,j+1) - 1
           &&  1 <= j+1 <= n)
    ASSERT(ForAll (k=1; k<j) (A | j -> A[j-1] + power(2,j))[k] == power(2,k+1) - 1
           && (A | j -> A[j-1] + power(2,j))[j] == power(2,j+1) - 1
           &&  1 <= j+1 <= n)
    ASSERT(ForAll (k=1; k<j+1) (A | j -> A[j-1] + power(2,j))[k] == power(2,k+1) - 1
           &&  1 <= j+1 <= n)
    A[j] = A[j-1] + power(2,j);
```

```
    ASSERT(ForAll (k=1; k<j+1) A[k] == power(2,k+1) - 1  &&  1 <= j+1 <= n)
    j++;
    ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1  &&  1 <= j <= n)
}
ASSERT(ForAll (k=1; k<j) A[k] == power(2,k+1) - 1  &&  j >= n && 1 <= j <= n)
FACT(j >= n && 1 <= j <= n => j == n)
ASSERT(ForAll (k=1; k<n) A[k] == power(2,k+1) - 1  &&  j >= n && 1 <= j <= n)
ASSERT(ForAll (k=1; k<n) A[k] == power(2,k+1) - 1  &&  j >= n)
ASSERT(ForAll (k=1; k<n) A[k] == power(2,k+1) - 1)
```

The proof of total correctness is pretty straightforward and so left as an exercise for the interested reader.