Additional Verification Techniques

This material is from Chapter 4 in the textbook. The proof tableau scheme for for-loops is given in Section 4.1.

As an example we verify the partial correctness of the following:

```c
ASSERT(0 <= n <= max)
{
    int i;
    for (i=0; A[i] != x && i < n; i++)
    {
    }
    present = i<n;
}

ASSERT(present iff x in A[0:n-1])

Note that the code may not terminate normally if x does not occur in A[0:n-1]. Why?

As the loop invariant (denoted as I) we choose:
0<=i<=n && ForAll(k=0; k<i) x != A[k]

Using the scheme for for-loops, we must verify the following:

ASSERT(pre-condition)
i=0; /* initial assignment */
ASSERT(I) /*loop invariant*/
while( A[i] != x && i < n) {
    ASSERT(I && A[i] != x && i < n)
    /* for-loop has empty body*/
    i++;
    ASSERT(I)
}

ASSERT(I && !(A[i] != x && i < n))
```
present = i<n; /*final assignment*/

ASSERT(post-condition)

The complete construction is given in class. Here we have to be a little careful in how
the post-condition is established from the invariant and the negation of the loop condition.

Array component assignment rule

The notation \((A | I \mapsto E)\) refers to an array obtained from \(A\) by replacing the value at
position \(I\) by the value of the expression \(E\).

More formally,
\[
(A|I \mapsto E)[I'] = \begin{cases} 
E & \text{when } I' = I, \\
A[I'] & \text{when } I' \neq I.
\end{cases}
\]

Now the array component assignment rule can be written as:
\[
[Q](A \mapsto A') \{A[I] = E; \} \ Q
\]

where \(A'\) is \((A | I \mapsto E)\).

It has to be verified separately that the value of \(I\) is within the subscript range of the
array \(A\).

Example. We consider an array of even length.

The program should move elements from even numbered positions to a contiguous chunk
at the beginning, see Figure 1.

The specification for the program is as follows:

Interface: const int n;

Entry A[2n]; /*entries numbered 0,...,2n-1 */

Pre-condition: n >= 1 && A == A0
Post-condition: $\forall i=0; \ i<n \ A[i] == A[2i]$

On the basis of the post-condition we can select a suitable loop invariant and using it “derive” the program. (To be done in class.)