Chapter 11: The Bounded Linear Search Theorem.

In which we introduce a very useful searching algorithm.

Suppose we are given an array f[0..N) of int, where $\{1 \le N\}$, and a target value X, and we are asked to find the location of the leftmost X in f, i.e. the smallest index n where f.n = X. This time however, we have no guarantee that X is in f.

We begin as usual with a problem specification.

There are two possibilities, either X is there or X is absent. In the case where it is present the postcondition we want to achieve is the same as in the Linear Search

$$\text{Post1} : \langle \forall j : 0 \le j < n : f.j \ne X\rangle \wedge f.n = X$$

In the case where it is absent we could phrase the postcondition as

$$\text{Post2} : \langle \forall j : 0 \le j < N : f.j \ne X\rangle$$

But instead we choose to phrase it as follows

$$\text{Post2} : \langle \forall j : 0 \le j < n : f.j \ne X\rangle \wedge (n = N\text{-}1 \wedge f.n \ne X)$$

In this way both Post1 and Post2 contain exactly the same quantified expression. We can now combine them to give our overall postcondition

$$\text{Post} : \langle \forall j : 0 \le j < n : f.j \ne X\rangle \wedge (f.n = X \vee (n = N\text{-}1 \wedge f.n \ne X))$$

Recalling one of our theorems from Boolean Calculus $[X \vee (\neg X \wedge Y) \equiv X \vee Y]$ we can now simplify this to get

$$\text{Post} : \langle \forall j : 0 \le j < n : f.j \ne X\rangle \wedge (f.n = X \vee n = N\text{-}1)$$

Model problem domain.

* (0) C.n $\equiv$ $\langle \forall j : 0 \le j < n : f.j \ne X\rangle$, $0 \le n \le N$

Consider.

C.0

= {(0) in model }

$\langle \forall j : 0 \le j < 0: f.j \ne X\rangle$

= { empty range }

true

- (1) $C.0 \quad \equiv \quad true$

Consider

$$
\begin{array}{ll}
 & C.(n+1) \\
= & \quad \{(0) \text{ in model }\} \\
 & \langle \forall j : 0 \leq j < n+1 : f.j \neq X \rangle \\
= & \quad \{ \text{ split off } j = n \text{ term}\} \\
 & \langle \forall j : 0 \leq j < n : f.j \neq X \rangle \wedge f.n \neq X \\
= & \quad \{(0) \text{ in model}\} \\
 & C.n \wedge f.n \neq X
\end{array}
$$

- (2) $C.(n+1) \quad \equiv \quad C.n \wedge f.n \neq X \qquad , 0 \leq n < N$

We can now rewrite our postcondition as

$$\text{Post} : C.n \wedge (f.n = X \vee n = N\text{-}1)$$

Choose Invariants.

We choose as our invariants

$$P0: C.n$$
$$P1: 0 \leq n \leq N$$

Termination.

We note that

$$P0 \wedge P1 \wedge (f.n = X \vee n = N\text{-}1) \Rightarrow \text{Post}$$

Establish Invariants.

Our model (1) shows us that we can establish P0 by the assignment

$$n := 0$$

This also establishes P1.

Guard.

We choose our loop guard to be

B : $f.n \neq X \wedge n \neq N - 1$

Variant.

As our variant function we choose

$$K - n$$

Where $0 \leq K < N$ and K is the (as yet unknown) index of the leftmost occurrence of X.

Calculate Loop body.

Decreasing the variant by the assignment n := n+1 is a standard step and maintains P1. Let us se what effect it has on P0

```
    (n := n+1). P0
=           {textual substitution}
    C.(n+1)
=           {(2) above}
    C.n ∧ f.n ≠ X
=           {Remember P0 ∧ f.n ≠ X ∧ n ≠ N – 1 at start of loop body}
    true
```

Final program.

```
  n := 0
; do f.n ≠ X ∧ n ≠ N - 1→

          n := n+1

  od
{C.n ∧ (f.n = X ∨ n = N-1)}
```

When the loop terminates we can now decide which outcome has occurred and communicate this to the user by adding a simple if..fi as follows.

```
if f.n= X                → write('X found at position', n )
[] n = N-1 ∧ f.n ≠ X     → write('X is not in f')
fi
```

General Solution.

This problem is just one instance of a set of problems called the Bounded Linear Searches. We will now describe this family of problems and construct the generic solution.

Suppose we are given a finite, ordered domain, $f[\alpha..\beta)$ and a predicate Q defined on the elements of f. We are to determine whether Q holds true at at least one point in the domain. Of course there is the possibility that it may not hold anywhere.

Our postcondition is

$$\langle \forall j : \alpha \leq j < i : \neg Q.(f.j) \rangle \wedge (Q.(f.i) \vee i = \beta - 1)$$

As usual we develop our model

* (0) $C.i \equiv \langle \forall j : \alpha \leq j < i : \neg Q.(f.j) \rangle \quad , \alpha \leq i \leq \beta$

Appealing to the empty range and associativity we get the following theorems

Consider.

$$
\begin{array}{ll}
 & C.\alpha \\
= & \quad \{(0) \text{ in model }\} \\
 & \langle \forall j : \alpha \leq j < \alpha : \neg Q.(f.j) \rangle \\
= & \quad \{ \text{ empty range } \} \\
 & true
\end{array}
$$

\- (1) $C.\alpha \equiv true$

Consider

$$
\begin{array}{ll}
 & C.(i+1) \\
= & \quad \{(0) \text{ in model }\} \\
 & \langle \forall j : \alpha \leq j < i+1 : \neg Q.(f.j) \rangle \\
= & \quad \{ \text{ split off } j = i \text{ term}\} \\
 & \langle \forall j : \alpha \leq j < i : \neg Q.(f.j) \rangle \wedge \neg Q.(f.i) \rangle \\
= & \quad \{(0) \text{ in model}\} \\
 & C.n \wedge \neg Q.(f.i)
\end{array}
$$

\- (2) $C.(i+1) \equiv C.i \wedge \neg Q.(f.i) \quad , \alpha \leq i < \beta$

Rewrite postcondition in terms of model.

$$\text{Post} : C.i \wedge (Q.(f.i) \vee i = \beta - 1)$$

Choose Invariants.

We choose as our invariants

$$P0: C.i$$
$$P1: \alpha \leq i < \beta$$

Termination.

We note that

$$P0 \wedge P1 \wedge (Q.(f.i) \vee i = \beta - 1) \Rightarrow \text{Post}$$

Establish Invariants.
Our model (1) shows us that we can establish P0 by the assignment

$$i := \alpha$$

This also establishes P1.

Guard
We choose our loop guard to be

$$B : \neg Q.(f.i) \wedge i \neq \beta - 1$$

Variant.
As our variant function we choose

$$K - i$$

Where $\alpha \leq K < \beta$ and K is the (as yet unknown) index of the leftmost point where Q holds.

Calculate Loop body.

Decreasing the variant by the assignment i := i+1 is a standard step and maintains P1. Let us se what effect it has on P0

$$
\begin{array}{ll}
 & (i := i+1).\ P0 \\
= & \qquad \{\text{textual substitution}\} \\
 & C.(i+1) \\
= & \qquad \{(2)\ \text{above}\} \\
 & C.i \wedge \neg Q.(f.i) \\
= & \qquad \{\text{Remember } P0 \wedge \neg Q.(f.i) \wedge i \neq \beta - 1 \text{ at start of loop body}\} \\
 & \text{true}
\end{array}
$$

Finished Program.

So our finished program is

```
i := α
; do ¬Q.(f.i) ∧ i ≠ β - 1→

            i := i+1

  od
{C.i ∧ (Q.(f.i) ∨ i = β - 1)}
```

We can now determine whether Q holds anywhere and communicate this with the user by adding the following if..fi after the loop

```
if Q.(f.i)               → write('X found at position', i )
[] i = β -1 ∧ ¬Q.(f.i)   → write('X is not in f')
fi
```

This is called the Bounded Linear Search Theorem.