Constructing the Binary Search once more

There are several reasons for writing this note: firstly I found a way of being more explicit than before about the design considerations, and this I want to have recorded, secondly it is an experiment in presenting the evolution of a design.

For given X, positive N and ascending array A, the program should establish

R0: $\quad present \equiv \langle \exists i: 0 \leq i < N: A[i] = X \rangle$.

When the value X does occur in array A and the program therefore should see to it that present is set to <u>true</u>, it is hard to see how this can be justified without at the same time establishing <u>where</u> in the array A value X occurs, i.e., for some variable i the program will establish

(0) $\quad A[i] = X$,

but as intermediate target this is too strong, for no such i exists when X does not occur in array A. In the latter case we might have to be content with establishing

(1) $\quad A[i] < X < A[i+1]$,

i.e. identifying the two nearest neighbours of X. Since the algorithm must be able to cope with both presence and absence of X, we suggest the combination of (0) and (1)

R_1: $\quad A[i] \leq X < A[i+1]$

as a more realistic target. (Note that we have $R_1 \equiv (0) \vee (1)$ if $A[i] < A[i+1]$.)

Summarizing, we are considering a program of the form

(2) { C } ||[var i : int
; "establish R_1"
; { R_1 } present := $A[i] = X$
]] { R_0 } .

We introduced the as yet unknown precondition C to cater for the possibility that we won't be able to establish R_1 under all circumstances; a C different from true means that ours is still a partial design.

It stands to reason to expect that larger values of N might require larger computation times and that, therefore, our program contains a repetitive construct. Program (2) tells us that that repetition occurs in "establish R_1". We make the simplest assumption, viz. that R_1 is established

by the completion of that repetition, which means that we must choose an invariant that is a generalization of R1. Again we choose (in principle) the simplest device, viz. that of replacing a constant by a fresh variable, j say. Replacing the constant 1 by j would yield for the invariant the conjunct

$$A[i] \leq X < A[i+j]$$

and for the guard of the repetition the expression

$$j \neq 1 \quad .$$

Because R1 is really the conjunction of two conditions, it is preferable to replace the whole subexpression $i+1$ by j. This yields for the invariant P the conjunct(s)

(3) $$A[i] \leq X \ \wedge \ X < A[j]$$

and for the repetion the guard

(4) $$j \neq i+1 \quad .$$

Remark From the point of view of information theory, the choice just made is immaterial: if p, q, r satisfy $p+q=r$, the values of any two of them determines the value of the third. From the point of view of disentangling the argument, the choice is significant: now each variable is associated with its own con-

junct to be maintained. (End of Remark.)

We postpone the discussion of how to maintain (3) during the repetition and ask ourselves first how we can establish it at the initialization of i,j. Looking at R0 - the only place where specific subscript values have been mentioned - I can think of only one initialization, viz.

$$i,j := 0, N \quad .$$

which only establishes (3) provided we have

(5) $\quad A[0] \leq X \;\wedge\; X < A[N]$.

We meet this condition by defining C to be equal to (5). Summarizing, we are in the mean time considering a program of the form

```
(6)  {C} |[ var i: int
         ; |[ var j: int
           ; i,j := 0,N {P}
           ; do j ≠ i+1 →
                      {P ∧ j≠i+1} "adjust i,j" {P}
             od {P ∧ j=i+1, hence}
           ]| {R1}
         ; {R1} present := A[i]=X
         ]| {R0}
```

We now turn to the design of such an "adjust i,j" that we can assure termination of the

repetition, i.e. we are looking for a variant function t that is decreased by "adjust i,j" and is bounded from below. The fact that upon termination $j-i$ equals 1 while initially it equals N, which is at least 1 but is allowed to be much larger, suggests to try

(7) $$t = j-i \quad ,$$

and to include in P the conjunct

(8) $$i<j$$

so as to ensure that t is bounded from below.

Now we take a seemingly bold step: we propose for "adjust i,j" an inner block of the form

(9)
```
{i+1<j} ][ var h: int
        ; h:= ..... {Q}
        ; if ..... → {Qi} i:=h {i<j}
          [] ..... → {Qj} j:=h {i<j}
          fi
        ]] {i<j}
```

Remark I wrote "seemingly bold", and I think rightly so, for the only constraint implied by (9) is that a single execution of "adjust i,j" modifies either i or j, and since (3) presents independent constraints on i and j, we can live with that: invariance of (3) will not re-

quire a single execution of "adjust i,j" to modify both i and j. (End of Remark)

We now calculate Qi and Qj from the requirements that t decrease and remain positive

Qi

≡ { definitions of t and statement in question}

$[i:=h](j-i) < j-i \;\wedge\; [i:=h](i<j)$

≡ { substitution operator twice}

$j-h < j-i \;\wedge\; h<j$

≡ { algebra}

$i<h \;\wedge\; h<j$

Qj

≡ {definitions of t and statement in question}

$[j:=h](j-i) < j-i \;\wedge\; [j:=h](i<j)$

≡ { substitution operator twice}

$h-i < j-i \;\wedge\; i<h$

≡ { algebra}

$h<j \;\wedge\; i<h$,

i.e., we find for Qi and Qj the <u>same</u> predicate

(10) $\quad i<h \;\wedge\; h<j$.

In response to this observation we choose (10) for Q, the precondition for the alternative construct. We point out that since here i and j are at least 2 apart, an h satisfying (10)

exists. We leave to the reader the verification that, without any need to strengthen guards of the alternative construct, the conjunct $0 \leq i \wedge j \leq N$ can be added to the invariant P.

As far as i,j are considered, we have reached the program

```
(11) {0<N} |[ var i : int
           ; |[ var j : int
             ; i,j := 0,N {P}
             ; do i+1 ≠ j → {P ∧ i+1<j}
                 |[ var h : int
                 ; h := avg.i.j {P ∧ i<h<j}
                 ; if true → i := h {P}
                   [] true → j := h {P}
                   fi
                 ]| {P}
               od {P ∧ i+1=j}
           ]| ; present := A[i]=X
      ]|
```

with P equal to $0 \leq i \wedge i<j \wedge j \leq N$, and as only requirement that avg.i.j yields a value strictly between i and j. No matter what avg we choose, nor how the nondeterminacy of the alternative construct is resolved, the repetition terminates, creating the state $i+1=j$.

And now the time has come for the postponed discussion of how to maintain (3), that is

$$A[i] \leq X \;\wedge\; X < A[j] \quad ,$$

as part of the invariant P. The first conjunct can only be falsified by the only assignment to i, viz. $i := h$, but the Axiom of Assignment tells us that its preassertion $\{A[h] \leq X\}$ implies its postassertion $\{A[i] \leq X\}$. Strengthening the preceding guard to the precondition does the job. Similarly the assignment $j := h$ is prevented from falsifying P by guarding it by $X < A[h]$. In other words, it suffices for the invariance of (3) to replace in the above program the alternative construct by

$$\begin{array}{l} \underline{\text{if}}\ A[h] \leq X \rightarrow i := h \\ \;[]\ X < A[h] \rightarrow j := h \\ \underline{\text{fi}} \end{array} \quad ,$$

and to our joy we observe that the disjunction of the strengthened guards is still equal to $\underline{\text{true}}$, so we did not introduce the danger of abortion.

Because of $0 \leq i < h < j \leq N$, access to the array A is in the repetion restricted to $A[h]$ for $0 < h < N$; in the setting of "present" $A[0]$ may be accessed. Thus access is restricted to $A[i]$ for $0 \leq i < N$, the range of the quanti-

fication in R0, our problem statement. As a consequence we need not bother about $X<A[N]$, the second conjunct of (5), the proposed precondition C for the whole program, since nothing prevents us from making $X<A[N]$ true by definition, say by defining $A[N] = +\infty$. With that symbolic value the program is certain to establish R1 provided also (5)'s first conjunct $A[0] \leq X$ holds. But what if $X<A[0]$? Well, R1 may not be established, but the repetition terminates with $0 \leq i < N$ - because it does so independently of X and A - and then "present" is set to false as it should.

* * *

Well, that's it. The "experiment in presenting the evolution of a design", which was mentioned in the first paragraph, was abandoned, I'm afraid. I wanted to develop the program without recording all the successive versions as that involves so much rewriting, and thought I could do that by writing a program text with, in the assertions, names of predicates which would be developed off line. I more or less did so with P but halfway the text I felt that the question of how to simulate chalk and blackboard in print is a separate issue, and that is how it got abandoned.

I did like program (11), in particular before I had inserted the assignment to "present" at the end -this insertion was done later-, because it nicely shows how the i,j-manipulation -and in particular the termination- is independent of X and A. It is now hard to believe that I have seen binary searches whose termination depended on A being sorted or on how rounding took place in calculating avg.i.j ! Besides this very clear separation of concerns I liked the calculations of Qi and Qj which yield the same predicate, but with the conjuncts interchanged. It is no accident that the development concentrated on j-i, i.e. on its lower bound and on its decrease, for this is the only place where i and j occur in combination, all other conjuncts in P depending either on i or on j.

Austin, 13 November 1999

prof.dr Edsger W.Dykstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
USA