

PROGRAMMING LANGUAGES:

IMPERATIVE PROGRAM CONSTRUCTION

8. CASE STUDIES

Shin-Cheng Mu

Autumn Term, 2021

National Taiwan University and Academia Sinica

FASTER DIVISION

QUOTIENT AND REMAINDER

- Recall the problem:

```
con  $A, B : \text{Int}$   $\{0 \leq A \wedge 0 < B\}$   
var  $q, r : \text{Int}$   
?  
 $\{A = q \times B + r \wedge 0 \leq r < B\}$  .
```

- Recall: recognising the postcondition as a conjunction, we use $A = q \times B + r \wedge 0 \leq r$ as the invariant and $\neg (r < B)$ as the guard.

- The program we came up with:

```
q, r := 0, A
{A = q × B + r ∧ 0 ≤ r, bnd : r}
do B ≤ r → q := q + 1
    r := r - B
od
{A = q × B + r ∧ 0 ≤ r < B} .
```

- In each iteration of the loop, r is decreased by B .
- We can probably get a quicker program by decreasing r by ... $2 \times B$, when possible.
- What about decreasing r by $4 \times B$, $8 \times B$,... etc?

```
con A, B : Int {0 ≤ A ∧ 0 < B}
var q, r, b, k : Int
...
{0 ≤ k ∧ b = 2k × B ∧ A < b}
...
{A = q × b + r ∧ 0 ≤ r < b ∧
  0 ≤ k ∧ b = 2k × B, bnd : b}
do b ≠ B → ...od
{A = q × B + r ∧ 0 ≤ r < B}
```

THE PROGRAM

```
con  $A, B : \text{Int}$   $\{0 \leq A \wedge 0 < B\}$ 
var  $q, r, b, k : \text{Int}$ 

 $b, k := B, 0$ 
do  $b \leq A \rightarrow b, k := b \times 2, k + 1$  od
 $\{0 \leq k \wedge b = 2^k \times B \wedge A < b\}$ 
 $q, r := 0, A$ 
 $\{A = q \times b + r \wedge 0 \leq r < b \wedge$ 
   $0 \leq k \wedge b = 2^k \times B, \text{bnd} : b\}$ 
do  $b \neq B \rightarrow$ 
  if  $r < b / 2 \rightarrow q, b, k := q \times 2, b / 2, k - 1$ 
  |  $b / 2 \leq r \rightarrow q, b, k, r := q \times 2 + 1, b / 2,$ 
     $k - 1, r - b / 2$ 
  fi
od
 $\{A = q \times B + r \wedge 0 \leq r < B\}$ 
```

Kaldewaij presented the following alternative. Do you prefer this program?

```
con A, B : Int {0 ≤ A ∧ 0 < B}
var q, r, b, k : Int

b := B
do b ≤ A → b, k := b × 2, k + 1 od
q, r := 0, A
do b ≠ B →
  q, b, k := q × 2, b / 2, k - 1
  if r < b → skip
  | b ≤ r → q, r := q + 1, r - b
fi
od
{A = q × B + r ∧ 0 ≤ r < B}
```

- The program has the advantage that we do not need to have $b / 2$ in the guards.
- Note what the first assignment establishes:

$$\begin{aligned}
 &\{A = q \times b + r \wedge 0 \leq r < b \wedge \\
 &\quad 0 \leq k \wedge b = 2^k \times B \wedge b \neq B\} \\
 &q, b, k := q \times 2, b / 2, k - 1 \\
 &\{A = q \times b + r \wedge 0 \leq r < 2 \times b \wedge \\
 &\quad 0 \leq k \wedge b = 2^k \times B\}
 \end{aligned}$$

BINARY SEARCH REVISITED

BINARY SEARCH

- Given a sorted array of N numbers and a key, either locate the position where the key resides in the array, or report that the key does not present in the array, in $O(\log N)$ time.
- A possible spec:

```
con  $N, K$ : Int { $0 < N$ }  
con  $F$ : array [0.. $N$ ) of Int { $F$  ascending}  
var  $l, r$ : Int  
bsearch  
{ $F[l] = K \vee \dots$ } .
```

THE VAN GASTEREN-FEIJEN APPROACH

- Van Gasteren and Feijen pointed a surprising fact: binary search does not apply only to sorted lists!
- In fact, they believe that comparing binary search to searching for a word in a dictionary is a major educational blunder.
- Their binary search: let Φ be a predicate on two integers with some additional constraints to be given later:

```
con  $M, N : \text{Int}$   $\{M < N \wedge \Phi M N \wedge \dots\}$   
var  $l, r : \text{Int}$   
bsearch  
 $\{M \leq l < N \wedge \Phi l (l + 1)\}$  .
```

INVARIANT AND BOUND

- Invariant: $\Phi \ l \ r \wedge M \leq l < r \leq N$, loop guard: $l + 1 \neq r$.
- Initialisation: $l, r := M, N$.
- Bound: $r - l$.
- For any m such that $l < m < r$, we have $r - m < r - l$ and $m - l < r - l$. Therefore both $l := m$ and $r := m$ decrease the bound.

CONSTRUCTING THE LOOP BODY

- For $l := m$ we calculate.

$$\begin{aligned} & (\Phi \ l \ r \wedge M \leq l < r \leq N \mid) [l \setminus m] \\ & \equiv \Phi \ l \ m \wedge M \leq m < r \leq N \\ & \Leftarrow \Phi \ l \ m \wedge M \leq l < m < r \leq N . \end{aligned}$$

- That $l < m < r$ is our assumption. The leftover $\Phi \ l \ m$ gives rise to a guarded command: $\Phi \ l \ m \rightarrow l := m$.
- The case with $r := m$ is similar.

THE PROGRAM SKELETON

```
{M < N ∧ Φ M N}  
l, r := M, N  
{Φ l r ∧ M ≤ l < r ≤ N, bnd : r - l}  
do l + 1 ≠ r →  
  {... ∧ l + 2 ≤ r}  
  m := anything s.t. l < m < r  
  {... ∧ l < m < r}  
  if Φ m r → l := m  
  | Φ l m → r := m  
fi  
od  
{M ≤ l < N ∧ Φ i (i + 1)}
```

Note: $m := (l + r) / 2$ is a valid choice, thanks to the precondition that $l + 2 \leq r$.

CONSTRAINTS ON Φ

- But we need the if to be total.
- Therefore we demand a constraint on Φ :

$$\Phi \ l \ r \Rightarrow \Phi \ l \ m \vee \Phi \ m \ r, \text{ if } l < m < r. \quad (1)$$

- Some Φ satisfying (1) (for F of appropriate type):

- $\Phi \ l \ r = F[l] \neq F[r],$
- $\Phi \ l \ r = F[l] < F[r],$
- $\Phi \ l \ r = F[l] \leq A \wedge A \leq F[r],$
- $\Phi \ l \ r = F[l] \times F[r] \leq 0,$
- $\Phi \ l \ r = F[l] \vee F[r],$
- $\Phi \ l \ r = \neg (Q \ l) \wedge Q \ r.$

- Van Gasteren and Feijen believe that $\Phi \ l \ r = F[l] \neq F[r]$ is a

SEARCHING FOR A KEY

- The case $\Phi \ l \ r = \neg (Q \ l) \wedge Q \ r$ is worth special attention.
- Choose $Q \ i = K < F[i]$ for some K .
- Therefore $\Phi \ l \ r = F[l] \leq K < F[r]$.
- That constitutes the binary search we wanted!
- The postcondition: $M \leq l < N \wedge F[l] \leq K < F[l + 1]$.
- Note that we do *not* yet need F to be sorted!
- The algorithm gives you some l such that $F[l] \leq K < F[l + 1]$. If there are more than one such l , one is returned non-deterministically.

- That F is sorted comes in when we need to establish that there is at most one l satisfying the postcondition.
- That is, either $F[l] = K$, or $\neg \langle \exists i : M \leq i < N : F[i] = K \rangle$.

THE PROGRAM... OR A PART OF IT

- Let $\Phi \ l \ r = F[l] \leq K < F[r]$.
- Processing the array fragment $F[a..b]$:

```
l, r := a, b
{  $\Phi \ l \ r \wedge a \leq l < r \leq b, bnd : r - l$  }
do l + 1  $\neq$  r  $\rightarrow$ 
    m := (l + r) / 2
    if F[m]  $\leq$  K  $\rightarrow$  l := m
    | K < F[m]  $\rightarrow$  r := m
fi
od
{  $a \leq l < b \wedge F[l] \leq K < F[l + 1]$  }
```

- Note that $F[a]$ and $F[b]$ are never accessed.
- This program is not yet complete....

INITIALISATION

- But wait.. to apply the algorithm to the entire array, we need the precondition $\Phi \ 0 \ N$, that is $F[0] \leq K < F[N]$. Is that true? (We don't even have $F[N]$.)
- One can rule out cases when the precondition do not hold (and also deal with empty array). E.g.

```
if  $0 = N \rightarrow present := False$   
  |  $0 < N \rightarrow$   
    if  $K < F[0] \rightarrow present := False$   
    |  $F[N - 1] = K \rightarrow present, l := True, N - 1$   
    |  $F[0] \leq K < F[N - 1] \rightarrow$   
       $a, b := 0, N - 1$   
      program above  
       $p := f \ l = K$   
fi
```

PSEUDO ELEMENTS

- But there is a better way... introduce two pseudo elements!
- Let $F[-1] = -\infty$ and $F[N] = \infty$.
- Initially, $\Phi \ 0 \ N$ is satisfied.
- In the code, $F[-1]$ and $F[N]$ are never accessed. Therefore we do not actually have to represent them!
- We need to be careful interpreting the result, once the main loop terminates, however.

THE PROGRAM (1)

Let $\Phi \ l \ r = F[l] \leq K < F[r]$.

con $N, K : \text{Int} \ \{0 \leq N\}$

con $F : \text{array} [0..N) \text{ of } \text{Int} \ \{F \text{ ascending} \wedge$

$F[-1] = -\infty \wedge F[N] = \infty\}$

var $l, r : \text{Int}$

var $p : \text{Bool}$

$l, r := -1, N$

$\{\Phi \ l \ r \wedge -1 \leq l < r \leq N, \text{bnd} : r - l\}$

do $l + 1 \neq r \rightarrow$

$m := (l + r) / 2$

if $F[m] \leq K \rightarrow l := m$

| $K < F[m] \rightarrow r := m$

fi

od

$\{-1 \leq l < N \wedge F[l] \leq K < F[l + 1]\}$

THE PROGRAM (2)

$\{0 \leq l < N \wedge F[l] \leq K < F[l + 1]\}$

if $-1 = l \rightarrow p := \text{False}$

| $0 \leq l \rightarrow p := F[l] = K$

fi

$\{p = \langle \exists i : 0 \leq i < N : F[i] = K \rangle \wedge$

$p \Rightarrow F[l] = K\}$

- Kaldewaij derived an alternative program that introduces only $F[N] = \infty$ (but not $F[-1] = -\infty$), while requiring the array to be non-empty.
- The main loop is the same. It is only post-loop interpretation that is different.

A MORE COMMON PROGRAM

- Recall that Bentley proposed using binary search as an exercise.
- Bentley's solution can be rephrased below:

```
l, r, p := 0, N - 1, False  
do l ≤ r →  
    m := (l + r) / 2  
    if F[m] < K → l := m + 1  
    | F[m] = k → p := True; break  
    | K < F[m] → r := m - 1  
    fi  
od
```


A MORE COMMON PROGRAM

I'd like to derive it, but

- it is harder to formally deal with *break*.
 - Still, Bentley employed a semi-formal reasoning using a loop invariant to argue for the correctness of the program.
- To relate the test $F[m] < K$ to $l := m + 1$ we have to bring in the fact that F is sorted earlier.

COMPARISON

- The two programs do not solve exactly the same problem (e.g. when there are multiple K s in F).
- Is the second program quicker because it assigns l and r to $m + 1$ and $m - 1$ rather than m ?
 - $l := m + 1$ because $F[m]$ is covered in another case;
 - $r := m - 1$ because a range is represented differently.
- Is it quicker to perform an extra test to *return* early?
 - When K is not in F , the test is wasted.
 - Rolfe claimed that single comparison is quicker in average.
 - Knuth: single comparison needs $17.5 \lg N + 17$ instructions, double comparison needs $18 \lg N - 16$ instructions.