

Programming Languages:

Imperative Program Construction

8. Case Studies

Shin-Cheng Mu

Autumn Term, 2022

1 Faster Division

Quotient and Remainder

- Recall the problem:

```

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

```

- 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:

```

 $q, r := 0, A$ 
 $\{A = q \times B + r \wedge 0 \leq r, bnd : r\}$ 
do  $B \leq r \rightarrow q := q + 1$ 
     $r := r - B$ 
od
 $\{A = q \times B + r \wedge 0 \leq r < B\}$  .

```

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

1.1 Division in $O(\log B)$ Time

Strategy...

```

con  $A, B : Int \{0 \leq A \wedge 0 < B\}$ 
var  $q, r, b, k : Int$ 
...
 $\{0 \leq k \wedge b = 2^k \times B \wedge A < b\}$ 
...
 $\{A = q \times b + r \wedge 0 \leq r < b \wedge$ 
     $0 \leq k \wedge b = 2^k \times B, bnd : b\}$ 
do  $b \neq B \rightarrow \dots$  od
 $\{A = q \times B + r \wedge 0 \leq r < B\}$ 

```

Generating $2^k \times B$

- It is easy to satisfy $b = 2^k \times B \wedge A < b$.

```

 $b, k := B, 0$ 
do  $b \leq A \rightarrow b, k := b \times 2, k + 1$  od
 $\{0 \leq k \wedge b = 2^k \times B \wedge A < b\}$ 

```

- What are the loop invariant and the bound?
- Initialisation for the next loop easily follows:

```

 $\{0 \leq k \wedge b = 2^k \times B \wedge A < b\}$ 
 $q, r := 0, A$ 
 $\{A = q \times b + r \wedge 0 \leq r < b \wedge$ 
     $0 \leq k \wedge b = 2^k \times B\}$ 

```

Decreasing b

- What needs to be done before we decrement b by half?

$$\begin{aligned}
 & (A = q \times b + r \wedge 0 \leq r < b)[b \setminus b/2] \\
 & \equiv (A = q \times (b/2) + r \wedge 0 \leq r < b/2)
 \end{aligned}$$

- We can restore the invariant by $q := q \times 2$...

$$\begin{aligned} & (A = q \times (b / 2) + r \wedge 0 \leq r < b / 2) [q \setminus q \times 2] \\ & \equiv A = (q \times 2) \times (b / 2) + r \wedge 0 \leq r < b / 2 \\ & \Leftarrow A = q \times b + r \wedge 0 \leq r < b / 2 \wedge b = 2^k \times B \end{aligned}$$

- only if we already have $r < b / 2$!
- That gives us one guarded command:

$$r < b / 2 \rightarrow q, b, k := q \times 2, b / 2, k - 1$$

Decreasing b – The Other Case

- What about the case when $b / 2 \leq r < b$?
- The task is to find a substitution such that

$$\begin{aligned} & (A = q \times (b / 2) + r \wedge 0 \leq r < b / 2) [? \setminus ?] \\ & \Leftarrow A = q \times b + r \wedge b / 2 \leq r < b \wedge b = 2^k \times B \end{aligned}$$

- Comparing $0 \leq r < b / 2$ and $b / 2 \leq r < b$, one might want to try a substitution containing $[r \setminus r - b / 2]$.

$$\begin{aligned} & (0 \leq r < b / 2) [r \setminus r - b / 2] \\ & \equiv 0 \leq r - b / 2 < b / 2 \\ & \Leftarrow b / 2 \leq r < b \wedge b = 2^k \times B . \end{aligned}$$

- Consider the former half of the expression:

$$\begin{aligned} & (A = q \times (b / 2) + r) [r \setminus r - b / 2] \\ & \equiv A = q \times (b / 2) + r - b / 2 \\ & \equiv A = (q - 1) \times (b / 2) + r . \end{aligned}$$

- Applying $[q \setminus q \times 2 + 1]$ gives us back $A = q \times b + r$.
- Therefore, another guarded command:

$$\begin{aligned} & b / 2 \leq r \rightarrow q, b, k, r := \\ & \quad q \times 2 + 1, b / 2, k - 1, r - b / 2 \end{aligned}$$

The Program

```

con  $A, B : Int \{0 \leq A \wedge 0 < B\}$ 
var  $q, r, b, k : Int$ 
 $b, k := B, 0$ 
do  $b \leq A \rightarrow b, k := b \times 2, k + 1$  od
 $\{0 \leq k \wedge b = 2^k \times B \wedge A < b\}$ 
 $q, r := 0, A$ 
 $\{A = q \times b + r \wedge 0 \leq r < b \wedge$ 
 $\quad 0 \leq k \wedge b = 2^k \times B, bnd : b\}$ 
do  $b \neq B \rightarrow$ 
 $\quad$  if  $r < b / 2 \rightarrow q, b, k := q \times 2, b / 2, k - 1$ 
 $\quad \mid b / 2 \leq r \rightarrow q, b, k, r := q \times 2 + 1, b / 2,$ 
 $\quad \quad \quad k - 1, r - b / 2$ 
 $\quad$  fi
od
 $\{A = q \times B + r \wedge 0 \leq r < B\}$ 

```

1.2 Alternative Programs

Existential Quantification

- The variable k is used in the proofs, but not needed for computing the output.
- One can remove k and use existential quantification in the assertions instead.

```

con  $A, B : Int \{0 \leq A \wedge 0 < B\}$ 
var  $q, r, b : Int$ 
 $b := B$ 
do  $b \leq A \rightarrow b := b \times 2$  od
 $\{\langle \exists k : 0 \leq k : b = 2^k \times B \rangle \wedge A < b\}$ 
 $q, r := 0, A$ 
 $\{A = q \times b + r \wedge 0 \leq r < b \wedge$ 
 $\quad \langle \exists k : 0 \leq k : b = 2^k \times B \rangle, bnd : b\}$ 
do  $b \neq B \rightarrow$ 
 $\quad$  if  $r < b / 2 \rightarrow q, b := q \times 2, b / 2$ 
 $\quad \mid b / 2 \leq r \rightarrow q, b, r := q \times 2 + 1, b / 2,$ 
 $\quad \quad \quad r - b / 2$ 
 $\quad$  fi
od
 $\{A = q \times B + r \wedge 0 \leq r < B\}$ 

```

- The variable k is called a “ghost variable” in Kaldewaij [?].
- We can introduce k and remove it later. Or we can deal with existential quantification in the proofs. Which style do you prefer?

Alternative Program

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

```

con  $A, B : Int \{0 \leq A \wedge 0 < B\}$ 
var  $q, r, b, k : Int$ 
 $b, k := B, 0$ 
do  $b \leq A \rightarrow b, k := b \times 2, k + 1$  od
 $q, r := 0, A$ 
do  $b \neq B \rightarrow$ 
   $q, b, k := q \times 2, b / 2, k - 1$ 
  if  $r < b \rightarrow skip$ 
  |  $b \leq r \rightarrow q, r := q + 1, r - b$ 
  fi
od
 $\{A = q \times B + r \wedge 0 \leq r < B\}$ 

```

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

$$\{A = q \times b + r \wedge 0 \leq r < b \wedge 0 \leq k \wedge b = 2^k \times B \wedge b \neq B\}$$

$$q, b, k := q \times 2, b / 2, k - 1$$

$$\{A = q \times b + r \wedge 0 \leq r < 2 \times b \wedge 0 \leq k \wedge b = 2^k \times B\}$$

A Historical Note

- The correctness of the **if** in the loop was actually a key example in Dahl [?], one of the earliest book on *structured programming*:

```

 $\{0 \leq r < b\}$ 
 $b := b / 2$ 
if  $r < b \rightarrow skip$ 
  |  $b \leq r \rightarrow r := r - b$ 
  fi
 $\{0 \leq r < b\}$ 

```

- Now we can prove its correctness by routine symbolic manipulation.
- In Dahl [?], Dijkstra needed about one page of textual proof. It shows how much symbolic reasoning has advanced since then.

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

```

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

```

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.

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

- 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 l (l + 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):

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

- Van Gasteren and Feijen believe that $\Phi \ l \ r \equiv F[l] \neq F[r]$ is a better example when explaining binary search.

2.2 Searching for a Key

- The case $\Phi \ l \ r \equiv \neg (Q \ l) \wedge Q \ r$ is worth special attention.
- Choose $Q \ i \equiv K < F[i]$ for some K .
- Therefore $\Phi \ l \ r \equiv F[l] \leq K < F[r]$.
- 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.

Sortedness

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

```

l, r := a, b
{Φ l r ∧ a ≤ l < r ≤ b, bnd : r - l}
do l + 1 ≠ r →
  m := (l + r) / 2
  if F[m] ≤ K → l := m
  | K < F[m] → r := m
fi
od
{a ≤ l < b ∧ F[l] ≤ 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  $0 = N \rightarrow p := False$ 
|  $0 < N \rightarrow$ 
  if  $K < F[0] \rightarrow p := False$ 
  |  $F[N - 1] = K \rightarrow p, l := True, N - 1$ 
  |  $F[0] \leq K < F[N - 1] \rightarrow$ 
     $a, b := 0, N - 1$ 
    program above
     $p := F[l] = K$ 
  fi
fi

```

- where p is *True* iff. K presents in F .

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  $N, K : Int \ \{0 \leq N\}$ 
con  $F : \text{array} \ [0..N) \ \text{of} \ Int \ \{F \text{ ascending} \wedge$ 
   $F[-1] = -\infty \wedge F[N] = \infty\}$ 
var  $l, m, r : Int$ 
var  $p : Bool$ 
 $l, r := -1, N$ 
 $\{ \Phi \ l \ r \wedge -1 \leq l < r \leq N, bnd : r - l \}$ 
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)

```

 $\{-1 \leq l < N \wedge F[l] \leq K < F[l + 1]\}$ 
if  $-1 = l \rightarrow p := False$ 
|  $0 \leq l \rightarrow p := F[l] = K$ 
fi
 $\{p = \langle \exists i : 0 \leq i < N : F[i] = K \rangle \wedge$ 
   $p \Rightarrow F[l] = K\}$ 

```

Alternative Program

- Kaldewaij [?, Sec. 6.3] 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.

2.3 Searching with Premature Return

A More Common Program

- Recall that Bentley [?, pp. 35-36] proposed using binary search as an exercise.
- Bentley's solution can be rephrased below:

```

 $l, r, p := 0, N - 1, False$ 
do  $l \leq r \rightarrow$ 
   $m := (l + r) / 2$ 
  if  $F[m] < K \rightarrow l := m + 1$ 
  |  $F[m] = K \rightarrow p := True; break$ 
  |  $K < F[m] \rightarrow r := m - 1$ 
  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 [?, Exercise 23, Section 6.2.1]: single comparison needs $17.5 \lg N + 17$ instructions, double comparison needs $18 \lg N - 16$ instructions.