PROGRAMMING LANGUAGES: IMPERATIVE PROGRAM CONSTRUCTION 7. LOOP CONSTRUCTION III: USING ASSOCIATIVITY

Shin-Cheng Mu Autumn, 2024

National Taiwan University and Academia Sinica

GENERAL USE OF ASSOCIATIVITY

TAIL RECURSION

• A function *f* is *tail recursive* if it looks like:

 $fx = hx, \quad \text{if } bx; \\ fx = f(gx), \quad \text{if } \neg(bx).$

 Tail recursive functions can be naturally computed in a loop. To derive a program that computes *f X* for given *X*:

```
con X; var r, x;
x := X
\{f x = f X\}
do \neg (b x) \rightarrow x := g x od
r := h x
\{r = f X\}
```

provided that the loop terminates.

USING ASSOCIATIVITY

- · What if the function to be computed is not tail recursive?
- Consider function *k* such that:

 $k x = a, \qquad \text{if } b x; \\ k x = h x \oplus k (g x), \qquad \text{if } \neg (b x).$

where \oplus is associative with identity *e*.

- Note that *k* is not tail recursive.
- Goal: establish r = k X for given X.
- Trick: use an invariant $r \oplus k x = k X$.
 - 'computed' \oplus 'to be computed' = k X.
 - Strategy: keep shifting stuffs from right hand side of \oplus to the left, until the right is *e*.

CONSTRUCTING THE LOOP BODY

If $h \times holds$: $r \oplus k x = k X$ $\equiv \{bx\}$ $r \oplus a = k X$ Otherwise: $r \oplus k x = k X$ $\equiv \{ \neg (bx) \}$ $r \oplus (h x \oplus k (q x)) = k X$ $\equiv \{ \oplus \text{ associative } \}$ $(r \oplus h x) \oplus k (g x) = k X$

 $\equiv (r \oplus k x = k X)[r, x \setminus r \oplus h x, g x].$

con X; var r, x;

$$r, x := e, X$$

$$\{r \oplus k \ x = k \ X\}$$
do $\neg (b \ x) \rightarrow r, x := r \oplus h \ x, g \ x \ od$

$$\{r \oplus a = k \ X\}$$

$$r := r \oplus a$$

$$\{r = k \ X\}$$

if the loop terminates.

EXAMPLE: EXPONENTATION

EXPONENTATION AGAIN

• Consider again computing A^B .

 $con A, B : Int \{0 \leq B\}$ var r : Int ? $\{r = A^B\}$

Notice that:

$$\begin{aligned} x^0 &= & 1 \\ x^y &= & 1 \times (x \times x)^{y \text{ div } 2} & \text{ if even } y, \\ &= & x \times x^{y-1} & \text{ if odd } y. \end{aligned}$$

- How does it fit the pattern above? (Hint: *k* now has type $(Int \times Int) \rightarrow Int.$)
- To be concrete, let us look at this specialised case in more detail.

- To achieve $r = A^B$, introduce variables a, b and choose invariant $r \times a^b = A^B$.
- To satisfy the invariant, initialise with r, a, b := 1, A, B.
- If b = 0 we have $r = A^{B}$. Therefore the strategy would be use *b* as bound and decrease *b*.

• How to decrease b? One might try b := b - 1. We calculate:

$$(r \times a^b = A^B)[b \setminus b - 1]$$

= $r \times a^{b-1} = A^B$.

• To fullfill the spec below

$$\{r \times a^{b} = A^{B}\}$$

r := ?
$$\{r \times a^{b-1} = A^{B}\}$$

One may choose $r := r \times a$.

• That results in the program (omitting the assertions):

```
con A, B : Int \{0 \leq B\}
var r, a, b : Int
r, a, b := 1, A, B
do b \neq 0 \rightarrow r := r \times a; b := b - 1 od
\{r = A^B\}
```

• This program use *O*(*B*) multiplications. But we wish to do better this time.

TRY TO DECREASE FASTER

• Or, we try to decrease *b* faster by halfing it (let (/) denote integer division).

 $(r \times a^b = A^B)[b \setminus b / 2]$ = $r \times a^{b/2} = A^B$.

How to fullfill the spec below?

$$\{r \times a^{b} = A^{B}\}$$
?
$$\{r \times a^{b/2} = A^{B}\}$$

• If we choose $a := a \times a$:

$$(r \times a^{b/2})[a \setminus a \times a]$$

= $r \times (a \times a)^{b/2}$
= $r \times (a^2)^{b/2}$
= $r \times a^{2 \times (b/2)}$
= $\{even b\}$

- But wait! For the last step to be valid we need even b!
- That means the program fragment has to be put under a guarded command:

```
even b \rightarrow

{r \times a^{b} = A^{B} \wedge even b}

a := a \times a

{r \times a^{b/2} = A^{B}}

b := b / 2

{r \times a^{b} = A^{B}}
```

· For that we need to introduce an if in the loop body.

FAST EXPONENTIATION

• We can put the b := b - 1 choice under an *odd* b guard, resulting in the following program:

```
con A, B : Int \{0 \leq B\}
var r, a, b : Int
r, a, b := 1, A, B
\{r \times a^b = A^B \land 0 \leq b, bnd : b\}
do b \neq 0 \rightarrow
   if odd b \rightarrow r := r \times a
                        h := h - 1
        even b \rightarrow a := a \times a
                        b := b / 2
   fi
od
\{r = A^B\}
```

This program uses O(1, D) multiplications

12 / 14

FAST EXPONENTIATION

od

- There is no reason, however, that you have to put the b := b 1 choice under an *odd b* guard.
- You might come up with something like this:

```
con A, B : Int \{0 \leq B\}
var r, a, b : Int
r, a, b := 1, A, B
\{r \times a^b = A^B \land 0 \leq b, bnd : b\}
do b \neq 0 \rightarrow
   r := r \times a
   b := b - 1
   if True \rightarrow skip
        even b \rightarrow a := a \times a
                       b := b / 2
   fi
```

- How do we construct branches?
- If a program fragment needs a side condition to work, we know that we need a guard.
- We keep constructing branches until the disjunction of all the guards can be satisfied.