Programming Languages:
Imperative Program Construction
7. Loop Construction III: Using Associativity

Shin-Cheng Mu

Autumn Term, 2021

National Taiwan University and Academia Sinica

General Use of Associativity

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

$$
\begin{aligned}
f\,x &= h\,x, && \text{if } b\,x;\\
f\,x &= f\,(g\,x), && \text{if } \neg(b\,x).
\end{aligned}
$$

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

$$
\begin{aligned}
&\mathbf{con}\ X;\ \mathbf{var}\ r, x;\\
&x := X\\
&\{f\,x = f\,X\}\\
&\mathbf{do}\ \neg(b\,x) \rightarrow x := g\,x\ \mathbf{od}\\
&r := h\,x\\
&\{r = f\,X\}
\end{aligned}
$$

provided that the loop terminates.

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

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

  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.

If $b\ x$ holds:

$$
\begin{aligned}
& r \oplus k\ x = k\ X \\
\equiv \quad & \{\ b\ x\ \} \\
& r \oplus a = k\ X.
\end{aligned}
$$

Otherwise:

$$
\begin{aligned}
& r \oplus k\ x = k\ X \\
\equiv \quad & \{\ \neg(b\ x)\ \} \\
& r \oplus (h\ x \oplus k\ (g\ x)) = k\ X \\
\equiv \quad & \{\ \oplus \text{ associative } \} \\
& (r \oplus h\ x) \oplus k\ (g\ x) = k\ X \\
\equiv \quad & (r \oplus k\ x = k\ X)[r, x \backslash r \oplus h\ x, g\ x].
\end{aligned}
$$

$$
\begin{array}{l}
\textbf{con}\ X;\ \ \textbf{var}\ r, x; \\
\\
r, x := e, X \\
\{r \oplus k\ x = k\ X\} \\
\textbf{do}\ \neg(b\ x) \rightarrow r, x := r \oplus h\ x, g\ x\ \textbf{od} \\
\{r \oplus a = k\ X\} \\
r := r \oplus a \\
\{r = k\ X\}
\end{array}
$$

if the loop terminates.

Example: Exponentation

- Consider again computing A^B.

$$
\begin{array}{l}
\mathbf{con}\ A, B : Int\ \{0 \leqslant B\} \\
\mathbf{var}\ r : Int \\
\ ? \\
\{r = A^B\}
\end{array}
$$

- Notice that:

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

- How does it fit the pattern above? (Hint: k now has type $(Int \times Int) \to Int$.)
- 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:

$$\begin{aligned} &(r \times a^b = A^B)[b \backslash b - 1] \\ = \; & r \times a^{b-1} = A^B \quad . \end{aligned}$$

- To fullfill the spec below

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

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

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

$$
\begin{array}{l}
\textbf{con } A, B : Int\ \{0 \leqslant B\} \\
\textbf{var } r, a, b : Int \\
r, a, b := 1, A, B \\
\textbf{do } b \neq 0 \rightarrow r := r \times a; b := b - 1 \textbf{ od} \\
\{r = A^B\}
\end{array}
$$

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

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

$$\begin{aligned} & (r \times a^b = A^B)[b \backslash b \mathbin{/} 2] \\ = \; & r \times a^{b/2} = A^B \quad . \end{aligned}$$

- How to fullfill the spec below?

$$\begin{aligned} & \{r \times a^b = A^B\} \\ & ? \\ & \{r \times a^{b/2} = A^B\} \end{aligned}$$

- If we choose $a := a \times a$:

$$\begin{aligned} & (r \times a^{b/2})[a \backslash a \times a] \\ = \; & r \times (a \times a)^{b/2} \\ = \; & r \times (a^2)^{b/2} \\ = \; & r \times a^{2 \times (b/2)} \\ = \; & \quad \{ \mathit{even}\ b \} \end{aligned}$$

- 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:

$$
\begin{array}{l}
\mathit{even}\ b \rightarrow \\
\quad \{r \times a^b = A^B \wedge \mathit{even}\ b\} \\
\quad a := a \times a \\
\quad \{r \times a^{b/2} = A^B\} \\
\quad b := b \;/\; 2 \\
\quad \{r \times a^b = A^B\}
\end{array}
$$

- 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:

$$
\begin{array}{l}
\textbf{con } A, B : \mathit{Int}\ \{0 \leqslant B\} \\
\textbf{var } r, a, b : \mathit{Int} \\
r, a, b := 1, A, B \\
\{r \times a^b = A^B \wedge 0 \leqslant b, \mathit{bnd} : b\} \\
\textbf{do } b \neq 0 \rightarrow \\
\quad \textbf{if} \quad \mathit{odd}\ b \rightarrow r := r \times a \\
\qquad\qquad\qquad\quad\ b := b - 1 \\
\qquad\ \mid \mathit{even}\ b \rightarrow a := a \times a \\
\qquad\qquad\qquad\quad\ b := b \,/\, 2 \\
\quad \textbf{fi} \\
\textbf{od} \\
\{r = A^B\}
\end{array}
$$

This program uses $O(\log B)$ multiplications

- 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:

$$
\begin{array}{l}
\textbf{con } A, B : \mathit{Int}\ \{0 \leqslant B\} \\
\textbf{var } r, a, b : \mathit{Int} \\
r, a, b := 1, A, B \\
\{r \times a^b = A^B \wedge 0 \leqslant b, \mathit{bnd} : b\} \\
\textbf{do } b \neq 0 \rightarrow \\
\quad r := r \times a \\
\quad b := b - 1 \\
\quad \textbf{if } \ \mathit{True} \ \rightarrow \mathit{skip} \\
\quad \quad \mid \mathit{even}\ b \rightarrow a := a \times a \\
\quad \quad \quad \quad \quad \quad b := b \mathbin{/} 2 \\
\quad \textbf{fi} \\
\textbf{od}
\end{array}
$$

Side Note: Constructing Branches

- 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.