Feijen's Table of Cubes Problem

CSci 550: Program Semantics & Derivation

H. Conrad Cunningham
cunningham@cs.olemiss.edu

Software Architecture Research Group
Department of Computer and Information Science
University of Mississippi
201 Weir Hall
University, Mississippi 38677 USA

November 2005

Problem Description

Requirement: Print cubes of first N natural numbers

Constraints: Cannot use an array.
Cannot use exponentiation or multiplication operators
Can only print one integer at a time

Note: For computing scientists, natural numbers are the integers 0, 1, 2, ...

Program Specification

(Hoare triple) $\{ Q \}$ Program $\{ R \}$
(Precondition) $Q : N \geq 0$

(Postcondition) $R : (\forall\, i : 0 \leq i < N : i^3 \text{ printed})$

Program Derivation

Programming is a goal-directed activity

- Focus on postcondition

- Speculate on program structure

  Is a loop needed?

  Yes, since only one integer printed at time

  Structure loop using semantics; include abstract components

  $\{\ Q\ \}$
  initialization $\{\ P$, the loop invariant $\}$;
  do $B \rightarrow \{\ P \wedge B\ \}$
  progress toward goal while preserving invariant
  $\{\ P\ \}$
  od $\{\ P \wedge \neg B\ \}$
  $\{\ R\ \}$

- Refine abstract parts of program

- First focus on loop invariant P and guard B

  $P \wedge \neg B$ must establish (i.e., imply) R

  Manipulate R syntactically to find candidate P and B

  Apply heuristics

Applying Heuristic
Replace a Constant by a Variable

- Seek candidate loop invariant P and guard B by generalizing R
- Identify the constants in $(\forall\, i : 0 \leq i < N : i^3 \text{ printed})$
  Three obvious: 0, 3, N.
- Choose to replace N by new variable n
- Identify new postcondition $R' : (\forall\, i : 0 \leq i < n : i^3 \text{ printed}) \wedge n = N$
- Continue by matching $P \wedge \neg B$ against shape of R'

Applying Heuristic
Delete a Conjunct

- Match $P \wedge \neg B$ against R': $(\forall\, i : 0 \leq i < n : i^3 \text{ printed}) \wedge n = N$
- Take conjunct $n = N$ of R'; make its negation loop guard B
- Take remainder of R' as loop invariant P_0
- Add new invariant P_1 giving valid range of n

  (Invariants) $P_0 : (\forall\, i : 0 \leq i < n : i^3 \text{ printed})$

  $P_1 : 0 \leq n \leq N$

- Henceforth, let P represent conjunction of all loop invariants identified, e.g., $P_0 \wedge P_1$

Applying Heuristic
Continue Deleting a Conjunct

- Remember invariant P from previous slide

  (Invariants) $P_0 : (\forall\, i : 0 \leq i < n : i^3 \text{ printed})$

  $P_1 : 0 \leq n \leq N$

- Initialize n such that P holds at beginning of loop: $n := 0$

- Progress toward termination by incrementing n by 1

  Can only print one integer at a time

  Progress statement: $n := n + 1$

- Reestablish P in loop body

  $\{\ Q\ \}$
  $n := 0\ \{$ invariant P: $P_0 \wedge P_1\ \}$;
  do $n \neq N \rightarrow \{\ P \wedge n \neq N\ \}$
  $\quad print.(n^3)$;
  $\quad n := n + 1$
  $\quad \{\ P\ \}$
  od $\{\ P \wedge n = N\ \}$
  $\{\ R\ \}$

- Continue derivation to eliminate unwanted expression n^3

Applying Heuristic
Strengthening the Invariant

- Continue derivation to eliminate unwanted expression n^3

- Add new variable x for troublesome expression n^3

- Change program so x always has value n^3 at print

- Add new invariant P_2: $x = n^3$

- Initialize x to establish invariant at beginning

- Update x to reestablish invariant

$\{\ Q\ \}$
$n,\ x := 0,\ 0\ \{$ invariant P: $P_0 \wedge P_1 \wedge P_2\ \}$;
do $n \neq N \rightarrow \{\ P \wedge n \neq N\ \}$
$\quad$ *print*.x;
$\quad$ $n, x := n + 1, E$
$\quad$ $\{\ P\ \}$
od $\{\ P \wedge n = N\ \}$
$\{\ R\ \}$

- Calculate appropriate value for abstract expression E

  Invariant must hold after execution of assignment

  Use semantics of assignment to determine what must hold before

  Assignment must update x to have value $(n + 1)^3$
  (n denotes value beforehand)

  Before assignment, x has value n^3

Calculation of Unknown Expression

- Axiom of assignment: $\{\ R(w, v := W, V)\ \}\ w, v := W, V\ \{\ R\ \}$

  $R(w, v := W, V)$ denotes R with variables w and v replaced textually by expressions W and V

- To prove specification $\{\ Q\ \}\ w, v := W, V\ \{\ R\ \}$,
  show $Q \Rightarrow R(w, v := W, V)$

- Find E such that $\{\ P \wedge n \neq N\ \}\ n, x := n + 1, E\ \{\ P\ \}$ holds

  Focus on P_2, the only invariant affected by x

  That is, find E such that $P \wedge n \neq N \Rightarrow P_2(n, x := n + 1, E)$

  $$
  \begin{array}{ll}
  & P_2(n, x := n + 1, E) \\
  \equiv & \quad \langle \text{ textual substitution } \rangle \\
  & E = (n + 1)^3 \\
  \equiv & \quad \langle \text{ arithmetic } \rangle \\
  & E = n^3 + 3 * n^2 + 3 * n + 1 \\
  \equiv & \quad \langle\ x = n^3\ (P_2) \text{ holds beforehand } \rangle \\
  & E = x + 3 * n^2 + 3 * n + 1
  \end{array}
  $$

  $\{\ Q\ \}$
  $n,\ x := 0,\ 0\ \{$ invariant P: $P_0 \wedge P_1 \wedge P_2\ \}$;
  do $n \neq N \rightarrow \{\ P \wedge n \neq N\ \}$
  $\quad print.x$;
  $\quad n, x := n + 1, x + 3 * n^2 + 3 * n + 1$
  $\quad \{\ P\ \}$
  od $\{\ P \wedge n = N\ \}$
  $\{\ R\ \}$

- Continue to eliminate remaining exponentiation and multiplications

Another Strengthening

- Eliminate remaining exponentiation and multiplications

- Again strengthen the invariant

- Add new variable y for troublesome expression

  Add invariant, initialization, and update statements

  (Invariant) $P_3 : y = 3 * n^2 + 3 * n + 1$

$\{\ Q\ \}$
$n, x, y := 0, 0, 1\ \{$ invariant P: $P_0 \wedge P_1 \wedge P_2 \wedge P_3\ \}$;
do $n \neq N \rightarrow \{\ P \wedge n \neq N\ \}$
$\quad print.x$;
$\quad n, x, y := n + 1, x + y, F$
$\quad \{\ P\ \}$
od $\{\ P \wedge n = N\ \}$
$\{\ R\ \}$

$$
\begin{array}{ll}
 & P_3(n, x, y := n + 1, x + y, F) \\
\equiv & \quad \langle \text{ textual substitution } \rangle \\
 & F = 3 * (n + 1)^2 + 3 * (n + 1) + 1 \\
\equiv & \quad \langle \text{ arithmetic } \rangle \\
 & F = 3 * n^2 + 9 * n + 7 \\
\equiv & \quad \langle\ P_3\ \rangle \\
 & F = y + 6 * n + 6
\end{array}
$$

- Continue to eliminate unwanted multiplication operation

  Can rewrite as six additions, but let's go for elegance

Yet Another Strengthening

- Again strengthen invariant by adding new variable z for troublesome expression

  Add invariant, initialization, and update statements

  (Invariant) $P_4 : z = 6 * n + 6$

  $\{\ Q\ \}$
  $n, x, y, z := 0, 0, 1, 6\ \ \{$ invariant P: $P_0 \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4\ \}$;
  do $n \neq N \rightarrow \{\ P \wedge n \neq N\ \}$
  $\quad print.x$;
  $\quad n, x, y, z := n + 1, x + y, y + z, G$
  $\quad \{\ P\ \}$
  od $\{\ P \wedge n = N\ \}$
  $\{\ R\ \}$

  $\quad P_4(n, x, y, z := n + 1, x + y, y + z, G)$
  $\equiv \quad \langle$ textual substitution $\rangle$
  $\quad G = 6 * (n + 1) + 6$
  $\equiv \quad \langle$ arithmetic $\rangle$
  $\quad G = 6 * n + 12$
  $\equiv \quad \langle\ P_4\ \rangle$
  $\quad G = z + 6$

- Arrive at final program

  $\{\ Q\ \}$
  $n, x, y, z := 0, 0, 1, 6\ \{$ invariant P: $P_0 \wedge P_1 \wedge P_2 \wedge P_3 \wedge P_4\ \}$;
  do $n \neq N \rightarrow \{\ P \wedge n \neq N\ \}$
  $\quad print.x$;
  $\quad n, x, y, z := n + 1, x + y, y + z, z + 6$
  $\quad \{\ P\ \}$
  od $\{\ P \wedge n = N\ \}$
  $\{\ R\ \}$

Review of Derivation

- Stated precise specification consisting of precondition and postcondition

- Determined loop was necessary

- Manipulated postcondition using heuristics to arrive at basic structure for loop

  Guard $n \neq N$, progress statement $n := n + 1$, initialization $n := 0$

- Attempted to calculate values of abstract expressions using semantics and logic

- When needed, applied heuristic "strengthening the invariant", adding new variables

- Continued until program satisfied requirements

- Derived program that would have been difficult to find otherwise

  Logic, semantics, and heuristics provided systematic problem-solving method

- Formal proof of program is reverse of derivation

Excerpts from Dijkstra's *On the Cruelty of Really Teaching Computing Science*

Well, when all is said and done, the only thing computers can do for us is to manipulate symbols and produce the result of such manipulations. ...

But before a computer is ready to perform a class of meaningful manipulations—or calculations, if you prefer—we must write a program.

What is a program? Several answers are possible.

We can view the program as what turns the general-purpose computer into a special-purpose symbol manipulator, and it does so without the need to change a single wire. ...

I prefer to describe it the other way around. The program is an abstract symbol manipulator which can be turned into a concrete one by supplying a computer to it.

After all, it is no longer the purpose of programs to instruct our machines; these days, it is the purpose of machines to execute our programs.

So, we have to design abstract symbol manipulators. We all know what they look like. They look like programs or—to use somewhat more general terminology—usually rather elaborate formulae from some formal system. ...

[The] programmer ... has to derive that formula; he has to derive that program.

We know of only one reliable way of doing that, *viz.*, symbol manipulation.

And now the circle is closed. We construct our mechanical symbol manipulators by means of human symbol manipulation.

Hence, computing science is—and will always be—concerned with the interplay between mechanized and human symbol manipulation usually referred to as "computing" and "programming," respectively.

Teaching to unsuspecting youngsters the effective use of formal methods is one of the joys of life because it so extremely rewarding. Within a few months, they find their way into a new world with a justified degree of confidence that is radically novel for them; within a few months, their concept of intellectual culture has acquired a radically new dimension. To my taste and style, that is what education is about.