Derivation Examples

In this document, we present calculational derivations of some simple programs. These derivations are based on the derivations in [Coh90, Kal90, Gri87].

1 Integer Division

Specification:

$$\begin{array}{l}\textbf{con } x : \text{int}\,\{x \geq 0\}\\ \textbf{con } y : \text{int}\,\{y > 0\}\\ \textbf{var } q : \text{int}, r : \text{int}\\ \quad \textit{Integer Division}\\ \text{Postcondition} : 0 \leq r < y \wedge (q * y + r = x)\end{array}$$

The program should not contain division operation.

Step1

We treat the preconditions $x \geq 0$ and $y > 0$ as global invariants since the predicates involve only immutable variables. We apply the *Take conjuncts as invariant* heuristics and select $0 \leq r \wedge (q * y + r = x)$ as an invariant to arrive at the following program.

$$\begin{array}{l}\{true\}\\ \quad \textbf{unkprog}_2\\ \{0 \leq r \wedge q * y + r = x\}\\ \quad \textbf{while} \quad \{inv : 0 \leq r \wedge q * y + r = x\}\\ \qquad \neg\,(r < y) \rightarrow\\ \qquad\qquad \{inv \wedge \neg\,(r < y)\}\\ \qquad\qquad \textbf{unkprog}_3\\ \qquad\qquad \{inv\}\\ \quad \textbf{end}\\ \{0 \leq r < y \wedge q * y + r = x\}\end{array}$$

Step 2

We directly guess assignment $q, r := 0, x$ for $\textbf{unkprog}_2$ as it establishes the invariant at the entry of the loop.

Step 3

We envision an assignment $r, q := r - e', q'$ for $\textbf{unkprog}_3$ where e' is a metavariable such that $e' > 0$. Proof obligation for $\textbf{unkprog}_3 : r, q := r - e', q'$ is :

$$inv \wedge \neg(r < y) \Rightarrow inv(r, q := r - e', q') \wedge e' > 0$$

We assume the antecedent and manipulate the consequent to find expressions for the metavariables e' and q'.

$$
\begin{array}{ll}
& inv(r, q := r - e', q') \wedge e' > 0 \\
\equiv & \quad \{ \text{ Definition of } inv \text{; textual substitution } \} \\
& 0 \leq r - e' \wedge (q' * y + r - e' = x) \wedge e' > 0 \\
\equiv & \quad \{ \text{ Simplify } \} \\
& e' \leq r \wedge (q' * y + r - e' = x) \wedge e' > 0 \\
\equiv & \quad \{ \ y \leq r; \ y > 0; \text{ instantiate } e' \text{ as } y; \text{ simplify } \} \\
& q' * y + r - y = x \\
\equiv & \quad \{ \text{ Simplify } \} \\
& (q' - 1) * y + r = x \\
\equiv & \quad \{ \ q * y + r = x; \text{ instantiate } q' \text{ as } q + 1; \text{ simplify } \} \\
& true
\end{array}
$$

We have derived assignment $r, q := r - y, q + 1$ for $\mathbf{unkprog}_3$. The final derived program is:

$$
\begin{array}{l}
\{true\} \\
\quad q, r := 0, x \\
\{0 \leq r \wedge q * y + r = x\} \\
\quad \textbf{while} \quad \{inv : 0 \leq r \wedge q * y + r = x\} \\
\qquad \neg (r < y) \rightarrow \\
\qquad\qquad \{inv \wedge \neg (r < y)\} \\
\qquad\qquad r, q := r - y, q + 1 \\
\qquad\qquad \{inv\} \\
\quad \textbf{end} \\
\{0 \leq r < y \wedge q * y + r = x\}
\end{array}
$$

2 Table of cubes

We want to derive a program for constructing a table of cubes using only additive operations.

The problem can be specified as follows.

$$
\begin{array}{l}
\textbf{con } N : \text{int} \, \{N \geq 0\} \, ; \\
\textbf{var } c : \text{array } [0..N) \text{ of int}; \\
\qquad S \\
\big\{R : \big(\forall i : 0 \leq i < N : c[i] = i^3\big)\big\}
\end{array}
$$

Our task is to derive program S to establish the postcondition R. We are not allowed to use exponentiation or multiplication operation in the solution.

Step 1

We start by applying the heuristic of replacing the constant N by a fresh variable n and rewrite the postcondition as $P_0 \wedge (n = N)$ where P_0 is defined as follows.

$$P_0 : \ \big(\forall i : 0 \leq i < n : c[i] = i^3\big)$$

After rewriting the postcondition, we arrive at the following program.

$$
\begin{array}{l}
\textbf{con}\ N : \text{int}\,\{N \geq 0\}\,; \\
\textbf{var}\ c : \text{array}\ [0..N)\ \text{of int}; \\
\textbf{var}\ n : \text{int}; \\
\quad S \\
\{P_0 \wedge (n = N)\}
\end{array}
$$

Note that the new postcondition $P_0 \wedge (n = N)$ implies the original postcondition R.

Step 2

We now apply the well-known heuristic of *taking a conjunct as an invariant*. We introduce a while loop and select P_0 as an invariant and the negation of the remaining conjunct as the guard of the while loop. We follow the general guideline of adding bounds on the introduced variables by adding $P_1 : 0 \leq n \leq N$ as an additional invariant. We also initialize n with 0 to establish the invariants at the start of the loop and increment n inside the loop body.

$$
\begin{array}{l}
\textbf{var}\ n : \text{int}; \\
n := 0; \\
\textbf{while}\quad \{Inv : P_0 \wedge P_1\} \\
\quad n \neq N \rightarrow \\
\qquad S_1; \\
\qquad n := n + 1 \\
\textbf{end} \\
\{P_0 \wedge (n = N)\}
\end{array}
$$

Step 3

We can select S_1 to be $c[n] := n^3$ as it preserves the invariants. However, we are not allowed to use exponentiation. To eliminate exponentiation, we introduce fresh variable x, and rewrite our program as:

$$
\begin{array}{l}
\textbf{var}\ n : \text{int}; \\
n := 0; \\
\textbf{while}\quad \{Inv : P_0 \wedge P_1\} \\
\quad n \neq N \rightarrow \\
\qquad //\text{establish } P_2 : x = n^3 \\
\qquad c[n] := x; \\
\qquad n := n + 1; \\
\textbf{end} \\
\{P_0 \wedge (n = N)\}
\end{array}
$$

This program is correct provided $P_2 : x = n^3$ is a precondition of the statement $c[n] := x$. In order to establish P_2 as a precondition to $c[n] := x$, we

maintain P_2 as a loop invariant. After adding P_2 as an invariant we arrive at the program

$$
\begin{array}{l}
\textbf{var } n,\ x : \text{int}; \\
n,\ x := 0,\ 0; \\
\textbf{while} \quad \{Inv : P_0 \wedge P_1 \wedge P_2\} \\
\quad n \neq N \rightarrow \\
\qquad c[n] := x; \\
\qquad n,\ x := n+1,\ x'; \\
\textbf{end}
\end{array}
$$

where x' must be chosen to maintain P_2.

Step 4

The invariant $P2$ is available before the assignment $n, x := n+1, x'$ since the preceding statement $(c[n] := x)$ does not change its validity. We now calculate x' such that P_2 in preserved by the loop body:

$$
\begin{array}{ll}
 & wp((n,\ x := n+1,\ x'), P_2) \\
\equiv & \quad \{ \text{ Definition of } P_2 \text{ and assignment } \} \\
 & x' = (n+1)^3 \\
\equiv & \quad \{ \text{ Arithmetic } \} \\
 & x' = n^3 + 3*n^2 + 3*n + 1 \\
\equiv & \quad \{ P_2, \text{ to eliminate an exponentiation } \} \\
 & x' = x + 3*n^2 + 3*n + 1 \\
\equiv & \quad \left\{ \begin{array}{l} \text{Assume } P_3 : y = 3*n^2 + 3*n + 1 \text{ to eliminate} \\ \text{the exponentiation and the multiplications} \end{array} \right\} \\
 & x' = x + y
\end{array}
$$

During the calculation of x', to eliminate exponentiation and multiplications, we assumed that P_3 holds. This can be ensured by maintaining P_3 as a loop invariant. The following program is correct provided y' is chosen to maintain the invariance of P_3.

$$
\begin{array}{l}
\textbf{var } n,\ x,\ y : \text{int}; \\
n,\ x,\ y := 0,\ 0,\ 1; \\
\textbf{while} \quad \{Inv : P_0 \wedge P_1 \wedge P_2 \wedge P_3\} \\
\quad n \neq N \rightarrow \\
\qquad c[n] := x; \\
\qquad n,\ x,\ y := n+1,\ x+y,\ y'; \\
\textbf{end}
\end{array}
$$

Step 5

We now calculate x' such that P_3 in preserved by the loop body:

$$
\begin{array}{ll}
& wp((n,\ x,\ y := n+1,\ x+y,\ y'),\ P_3) \\
\equiv & \quad \{ \text{ Definitions of } P_3 \text{ and assignment } \} \\
& y' = 3*(n+1)^2 + 3*(n+1) + 1 \\
\equiv & \quad \{ \text{ Arithmetic} \} \\
& y' = 3*n^2 + 9*n + 7 \\
\equiv & \quad \{ P_3 \} \\
& y' = y + 6*n + 6
\end{array}
$$

After substituting $y + 6*n + 6$ for y in the program from the previous step, we arrive at the following program.

$$
\begin{array}{l}
\textbf{con } N : \text{int}\,\{N \geq 0\}\,; \\
\textbf{var } c : \text{array } [0..N) \text{ of int}; \\
\textbf{var } n,\ x,\ y : \text{int}; \\
n, x, y := 0, 0, 1; \\
\textbf{while} \quad \{Inv : P_0 \wedge P_1 \wedge P_2 \wedge P_3\} \\
\quad n \neq N \rightarrow \\
\qquad c[n] := x; \\
\qquad n,\ x,\ y := n+1,\ x+y,\ y+6*n+6 \\
\textbf{end}
\end{array}
$$

The term $6 * n$ can be easily rewritten using only the addition operation. We now have a linear-time solution for the problem.

3 Dutch National Flag

The specification of the *Dutch National Flag* program is

$$
\begin{array}{l}
\textbf{con } N : \text{int}\,\{N \geq 0\}\,; \\
\textbf{var } a : \text{array } [0..N) \text{ of } [red, white, blue]; \\
\textbf{var } r : \text{int}, w : \text{int}; \\
\textit{ Dutch National Flag} \\
R : \{Red(a, 0, r) \wedge White(a, r, w) \wedge Blue(a, w, N) \\
\qquad \wedge\ 0 \leq r \leq w \leq N\} \\
\text{where,} \\
\qquad Red(a, x, y) \triangleq (\forall\ i :\ x \leq i < y :\ a[i] = red)\,, \\
\qquad White(a, x, y) \triangleq (\forall\ i :\ x \leq i < y :\ a[i] = white)\,, \\
\qquad Blue(a, x, y) \triangleq (\forall\ i :\ x \leq i < y :\ a[i] = blue)
\end{array}
$$

where only array swap operations are allowed on the array a.

Step 1

The specification can be represented as the following annotated program.

$$
\begin{array}{l}
\textbf{con } N : \text{int} \\
\textbf{var } a : \text{array } [0..N) \text{ of } [red, white, blue]; \\
\textbf{var } r : \text{int}, w : \text{int}; \\
\hline
\text{Global Inv: } N \geq 0 \\
\hline
\{true\} \\
\quad \textbf{unkprog}_1 \\
\{Red(a,0,r) \wedge White(a,r,w) \wedge Blue(a,w,N) \\
\wedge\ 0 \leq r \leq w \leq N\}
\end{array}
$$

Step 2: Rewrite the postcondition

We replace the term w in $Blue(w, N)$ by a fresh variable b. We add bounds on b: $(w \leq b \leq N)$ and also add conjuct $b = w$ to the postcondition so that new postcondition implies the original postcondition. The program now becomes:

$$
\begin{array}{l}
\{true\} \\
\quad \textbf{var } b; \\
\quad \textbf{unkprog}_2 \\
\{Red(a,0,r) \wedge White(a,r,w) \wedge Blue(a,b,N) \\
\wedge\ 0 \leq r \leq w \leq N \wedge w \leq b \leq N \wedge b = w\}
\end{array}
$$

Step 3: Introduce while loop

We now apply the *take conjucts as invariants* heuristic and introduce a while loop with the following invariant and guard.

$$
\begin{array}{ll}
\text{Invariant } P: & Red(a,0,r) \wedge White(a,r,w) \wedge Blue(a,b,N)\ \wedge\ 0 \leq r \leq w \leq b \leq N \\
\text{Guard}: & b \neq w
\end{array}
$$

$$
\begin{array}{l}
\{true\} \\
\quad \textbf{unkprog}_3 \\
\{P\} \\
\quad \textbf{while} \quad \{inv : P\} \\
\qquad b \neq w \rightarrow \\
\qquad\qquad \{P\ \wedge\ b \neq w\} \\
\qquad\qquad \textbf{unkprog}_4 \\
\qquad\qquad \{P\} \\
\quad \textbf{end} \\
\{P\ \wedge\ (b = w)\}
\end{array}
$$

The invariant P can be easily established at the entry of the loop by the assignment $r,\ w,\ b := 0,\ 0,\ N$.

Step 4

The elements $a[w..b)$ are not yet inspected. We can choose to inspect $a[w]$ or $a[b-1]$ inside the loop body. Here we choose to inspect $a[w]$. We introduce an *if* construct to handle the three cases depending on the color of $a[w]$.

$$
\begin{array}{l}
\{true\} \\
\quad r, w, b := 0, 0, N \\
\{P\} \\
\quad \textbf{while} \quad \{inv : P\} \\
\qquad b \neq w \rightarrow \\
\qquad\qquad \{P \wedge b \neq w\} \\
\qquad\qquad \textbf{if} \\
\qquad\qquad\quad a[w] = red \rightarrow \textbf{unkprog}_5 \\
\qquad\qquad\quad a[w] = white \rightarrow \textbf{unkprog}_6 \\
\qquad\qquad\quad a[w] = blue \rightarrow \textbf{unkprog}_7 \\
\qquad\qquad \textbf{end} \\
\qquad\qquad \{P\} \\
\quad \textbf{end} \\
\{P \wedge b = w\}
\end{array}
$$

Step 5

The precondition of the loop body is

$$
\begin{array}{l}
Red(a, 0, r) \wedge White(a, r, w) \wedge Blue(a, b, N) \\
\wedge\ 0 \leq r \leq w < b \leq N
\end{array}
$$

We have following four segments in the array.

$$
\begin{array}{l}
a[0, r) : red \text{ elements} \\
a[r, w) : white \text{ elements} \\
a[w, b) : \text{uninspected elements} \\
a[b, N) : blue \text{ elements}
\end{array}
$$

Thinking of just the indices, for the case "$a[w] = red$", we will need to increment r as well as w; for the case "$a[w] = white$", only w needs to be incremented; and for the "$a[w] = blue$" case b needs to be decremented. Furthermore, it is clear that for the "a[w] = white" case, no other operation is required apart from the incrementing w. With this, we arrive at the follwing program.

$\{P \wedge b \neq w\}$
if
$a[w] = red \rightarrow$ **unkprog**$_8$
$r, w := r + 1, w + 1$
$a[w] = white \rightarrow w := w + 1$
$a[w] = blue \rightarrow$ **unkprog**$_9$
$b := b - 1$
end
$\{P\}$

Step 6

The program **unkprog**$_8$ with its precondition and postcondition is given below.

$$\{P \wedge b \neq w \wedge a[w] = red\}$$
$$\textbf{unkprog}_8$$
$$\{P(r, w := r + 1, w + 1)\}$$

The before-after predicate for this program is given below.

$$\begin{array}{l} Red(a, 0, r) \wedge White(a, r, w) \wedge Blue(a, b, N) \\ \wedge\ 0 \leq r \leq w < b \leq N \wedge (a[w] = red) \\ \Rightarrow \\ Red(a', 0, r + 1) \wedge White(a', r + 1, w + 1) \wedge Blue(a', b, N) \\ \wedge\ 0 \leq r + 1 \leq w + 1 \leq b \leq N \end{array}$$

In the above formula, a' denotes the value of array a in the postcondition. We assume the antecedent and manipulate the consequent in order to find value for a'.

$$\begin{array}{ll} & Red(a', 0, r + 1) \wedge White(a', r + 1, w + 1) \wedge Blue(a', b, N) \\ & \wedge\ 0 \leq r + 1 \leq w + 1 \leq b \leq N \\ \equiv & \quad \{\ 0 \leq r \leq w < b \leq N\ \} \\ & Red(a', 0, r + 1) \wedge White(a', r + 1, w + 1) \wedge Blue(a', b, N) \\ \equiv & \quad \{\ \text{Assume } a'[b, N) = a[b, N) \text{ and notice that } Blue(b, N)\} \\ & Red(a', 0, r + 1) \wedge White(a', r + 1, w + 1) \\ \equiv & \quad \{\ \text{Range Split}\ \} \\ & Red(a', 0, r) \wedge (a'[r] = red) \wedge White(a', r + 1, w) \wedge (a'[w] = white) \\ \equiv & \quad \{\ \text{Assume } a'[0, r) = a[0, r) \text{ and notice that } Red(0, r)\} \\ & (a'[r] = red) \wedge White(a', r + 1, w) \wedge (a'[w] = white) \\ \equiv & \quad \{\ \text{Assume } a'[r + 1, w) = a[r + 1, w) \text{ and notice that } White(r, w)\ \} \\ & (a'[r] = red) \wedge (a'[w] = white) \\ \equiv & \quad \{\ \text{Notice } a[r] = white \text{ and } a[w] = red\text{; Assume } a'[r] = a[w] \text{ and } a'[w] = a[r]\ \} \\ & true \end{array}$$

In the above calculation, we have assumed following predicates about a'.

$$
\begin{array}{l}
a'[0,r) = a[0,r) \\
a'[r] = a[w] \\
a'[r+1,w) = a[r+1,w) \\
a'[w] = a[r] \\
a'[b,N) = a[b,N)
\end{array}
$$

The above assumptions can be satisfied if we choose $\mathbf{unkprog}_8$ to be $Swap(a,r,w)$.

Similarly, we can derive $\mathbf{unkprog}_9$ to be $Swap(a,w,b-1)$ The final program is given below.

$$
\begin{array}{l}
\{true\} \\
\quad r,w,b := 0,0,N \\
\{P\} \\
\quad \textbf{while} \quad \{inv : P\} \\
\qquad b \neq w \rightarrow \\
\qquad\qquad \{P \wedge b \neq w\} \\
\qquad\qquad \textbf{if} \\
\qquad\qquad\quad a[w] = red \rightarrow Swap(a,r,w) \\
\qquad\qquad\qquad\qquad\qquad r,w := r+1, w+1 \\
\qquad\qquad\quad a[w] = white \rightarrow w := w+1 \\
\qquad\qquad\quad a[w] = blue \rightarrow Swap(a,w,b-1) \\
\qquad\qquad\qquad\qquad\qquad\quad b := b-1 \\
\qquad\qquad \textbf{end} \\
\qquad\qquad \{P\} \\
\quad \textbf{end} \\
\{P \wedge (b = w)\}
\end{array}
$$

References

[Coh90] Edward Cohen. *Programming in the 1990s - An Introduction to the Calculation of Programs.* Texts and Monographs in Computer Science. Springer, 1990.

[Gri87] David Gries. *The Science of Programming.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1987.

[Kal90] Anne Kaldewaij. *Programming: The Derivation of Algorithms.* Prentice-Hall, Inc., NJ, USA, 1990.