Finding Invariants

Part 1: Adding Parameters by Replacing Constants by Variables

CS 536: Science of Programming, Fall 2021

A. Why

• It is easier to write good programs and check them for defects than to write bad programs and then debug them.
• The hardest part of programming is finding good loop invariants.
• There are heuristics for finding them but no algorithms that work in all cases.

B. Objectives

At the end of this activity assignment you should

• Be able to how to generate possible invariants using “replace a constant by a variable” or more generally “add a parameter”.

C. Problems

1. What are the constants in the postcondition $x = \max(b[0], b[1], ..., b[n-1])$? Using the technique “replace a constant by a variable,” list the possible invariants for this postcondition. Also, what would the loop tests be? (Assume $n-1$ is a constant.)

2. Repeat, on the postcondition $x = n!$, where $n!$ is short for a function call $\text{product}(1, n)$.

3. Repeat, on the postcondition $\forall i. 0 \leq i < n \rightarrow b[i] = 3$.

4. Repeat, on the postcondition $\forall i. \forall j. 0 \leq i < K \land K \leq j < n \rightarrow b[i] < b[j]$. (Every value in $b[0...K-1]$ is < every value in $b[K...n-1]$.)
Solution to Practice 19 (Finding Invariants; Examples)

1. Certainly 0 is a constant; if we replace it by a variable $i$, we get
   \[
   \{ \text{inv } x = \max(b[i], ..., b[n-1]) \land 0 \leq i \leq n-1 \} \text{ while } i \neq 0 \text{ do } ...
   \]
   As a constant, $n-1$ seems better than just $n$ or 1 by themselves:
   \[
   \{ \text{inv } x = \max(b[0], ..., b[j]) \land 0 \leq j \leq n-1 \} \text{ while } j \neq n-1 \text{ do } ...
   \]
   If you want to treat just $n$ as a constant and replace it by a variable $j$, we get
   \[
   \{ \text{inv } x = \max(b[0], ..., b[j-1]) \land 1 \leq j \leq n \} \text{ while } j \neq n \text{ do } ...
   \]
   Similarly, if you want replace just the 1 in $n-1$ by with $j$, we get
   \[
   \{ \text{inv } x = \max(b[0], ..., b[n-j]) \land 1 \leq j \leq n \} \text{ while } j \neq 1 \text{ do } ...
   \]

2. We can replace $n$ by a variable and get
   \[
   \text{inv } x = i! \land 1 \leq i \leq n \}
   \text{ while } i \neq n \text{ do } ...
   \]
   We can replace 1 and get
   \[
   \{ \text{inv } x = j*(j+1)*...*n \land 1 \leq j \leq n \}
   \text{ while } j \neq 1 \text{ do } ...
   \]

3. For $\forall i. 0 \leq i < n \rightarrow b[i] = 3$ as the postcondition, we can replace 0 or $n$ or 3.
   Replace 0 by $k$:
   \[
   \{ \text{inv } 0 \leq k \leq n-1 \land \forall i. k \leq i < n \rightarrow b[i] = 3 \} \text{ while } k \neq 0 \text{ do } ...
   \]
   Replace $n$ by $k$
   \[
   \{ \text{inv } 0 \leq k \leq n \land \forall i. 0 \leq i < k \rightarrow b[i] = 3 \} \text{ while } k \neq n \text{ do } ...
   \]
   Replace 3 by $k$ (this doesn’t look useful)
   \[
   \{ \text{inv } \forall i. 0 \leq i < n \rightarrow b[i] = k \} \text{ while } k \neq 3 \text{ do } ...
   \]

4. For $\forall i. \forall j. 0 \leq i < K \land K \leq j < n \rightarrow b[i] < b[j]$, we have constants 0, $n$, the two occurrences of $K$.
   Replace 0 by $k$:
   \[
   \{ \text{inv } 0 \leq k < K \land \forall i. \forall j. k \leq i < K \land K \leq j < n \rightarrow b[i] < b[j] \}
   \text{ while } k \neq 0
   \]
   Replace left $K$ by $k$:
   \[
   \{ \text{inv } 0 \leq k < K \land \forall i. \forall j. 0 \leq i < k \land K \leq j < n \rightarrow b[i] < b[j] \}
   \text{ while } k \neq K
   \]
   Replace right $K$ by $k$:
   \[
   \{ \text{inv } K \leq k \leq n \land \forall i. \forall j. 0 \leq i < K \land k \leq j < n \rightarrow b[i] < b[j] \}
   \text{ while } k \neq K
   \]
   Replace $n$ by $k$:
   \[
   \{ \text{inv } K \leq k \leq n \land \forall i. \forall j. 0 \leq i < K \land K \leq j < K \rightarrow b[i] < b[j] \}
   \text{ while } k \neq n
   \]
You could argue that the ranges for $k$ could be $0 \leq k < n$, $0 \leq k < n$, $0 \leq k \leq n$, and $0 \leq k \leq n$ for the four cases above; it depends on knowing more about the context of the problem.