Finding Invariants

Part 2: Deleting Conjuncts; Adding Disjuncts

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

• Know how to generate possible invariants using the techniques “Drop a conjunct” and “Add a disjunct”.

C. Problems

1. Consider the postcondition \( x^2 \leq n < (x+1)^2 \), which is short for \( x^2 \leq n \land n < (x+1)^2 \). List the possible invariant/loop test combinations you can get for this postcondition using the technique “Drop a conjunct.”

2. Why is the technique “Drop a conjunct” a special case of “Add a disjunct”?

3. One way to view a search is as follows:
   
   \[
   \{ \text{inv found} \lor \text{not found} \}
   \]
   
   \[
   \text{while not found do}
   \]
   
   \[
   \quad \text{Remove something or somethings from the things to look at}
   \]
   
   \[
   \text{od}
   \]

For this problem, try to recast (a) linear search and (b) binary search of an array using this framework: What parts of that program correspond to “we have found it”, “we haven’t found it”, and “Remove something...”?
Solution to Activity 20 (Finding Invariants; Examples)

1. \{ inv \; n < (x+1)^2 \} \; \text{while} \; x^2 > n \ldots
   \{ inv \; x^2 \leq n \} \; \text{while} \; n \geq (x+1)^2 \ldots

2. Dropping a conjunct is like adding the difference between the dropped conjunct and the rest of the predicate. E.g., dropping $p_1$ from $p_1 \land p_2 \land p_3$ is like adding $(\neg p_1 \land p_2 \land p_3)$ to $(p_1 \land p_2 \land p_3)$.

3. (Rephrasing searches)
   a. We can rephrase linear search through an array with
      We have found it: \; k < n \land b[k] = x
      We haven’t found it: \; k < n \land b[k] \neq x
      Remove what we’re looking at from the things to look at: \; k := k+1
   b. We can rephrase binary search through an array with
      We have found it: \; R = L+1
      We haven’t found it: \; R > L+1
      Remove the left or right half from the things to look at: Either \; L := m \text{ or } R := m