Algorithms – CS-37000

Loop invariants

January 24, 2006

Instructor: László Babai

Loop invariants are critical tools for the proof of correctness of algorithms; they represent the "inductive step" in a proof by induction that the configuartion of the variables satisfies certain conditions throughout the algorithm.

To formalize this concept, we introduce some terminology.

Let $x_1, \ldots, x_m$ denote the variables on which an algorithm operates; let A_i be the domain of x_i (set of possible values of x_i). A *configuration* $a = (a_1, \ldots, a_m)$ is an assignment of values to each variable ($a_i \in A_i$). The set of all conceivable configurations is $\mathcal{C} = A_1 \times \ldots \times A_m$; we call $\mathcal{C}$ the *configuration space.* A *feasible* configuration is a configuration which can actually occur in the course of an execution of the algorithm. Note that in general, not all configurations are feasible.

Example: the variables in Dijkstra's algorithm are the priority queue L and for each vertex $i \in V$, the variables status(i), $c(i)$, and $p(i)$ (the current status, cost, and parent of vertex i), a total of $3n + 1$ variables where n is the number of vertices. The domain of status(i) is $\{$ white, grey, black $\}$; the domain of $c(i)$ is $\mathbb{R}^+ \cup \{\infty\}$ (the nonnegative reals and infinity); the domain of $p(i)$ is $V \cup \{ \text{NIL} \}$. The domain of L can be thought of as 2^V (the set of all subsets of V).

An example of an infeasible configuration that nevertheless belongs to the configuration space is a configuration where some vertex i belongs to the queue while status(i) $=$ black.

A *predicate* over $\mathcal{C}$ is a function $P : \mathcal{C} \to \{0, 1\}$ where 0 indicates "FALSE" and 1 indicates "TRUE." A *transformation* of $\mathcal{C}$ is a function $S : \mathcal{C} \to \mathcal{C}$.

If P is a predicate and $a \in \mathcal{C}$ a configuration then instead of writing $P(a) = 1$, we just write "$P(a)$," meaning "the statement $P(a)$ is TRUE"; i. e., the configuration a *satisfies* the predicate P. For $P(a) = 0$ we may write "$\neg P(a)$," meaning the negation of $P(a)$ holds, i. e., a does not satisfy P. In other words, P is false on a.

The effect of a sequence S of instructions in the code is a change of the values of the variables and therefore S can be thought of as a *transformation* $S : \mathcal{C} \to \mathcal{C}$.

We are now ready to define the concept of loop-invariants.

Definition. Let P and Q be predicates over the configuration space and let S be a sequence of instructions, viewed as a transformation of the configuration space. Consider the loop

$$\textbf{while } P \textbf{ do } S.$$

We call Q a **loop-invariant** for this loop if for all configurations a it is true that

$$(\forall a \in \mathcal{C})(\text{ if } P(a) \,\&\, Q(a) \text{ then } Q(S(a))).$$

In other words, whenever a configuration $a \in \mathcal{C}$ satisfies the loop condition P and the predicate Q, the new configuration $S(a)$ obtained by executing the sequence S of instructions again satisfies Q.

Most important here is the quantifier $(\forall a \in \mathcal{C})$. The inference "if $P(a) \,\&\, Q(a)$ then $Q(S(a))$" must be valid even if a is not a feasible configuration. The power of loop-invariants comes from this feature; no hidden assumptions are permitted.

The situation has some similarity with chess puzzles: when showing that a certain configuration leads to checkmate in two moves, you do not investigate whether or not the given configuration could arise in an actual game.

PRACTICE QUESTIONS

Dijkstra's algorithm consists of iterations of a single "**while**" loop. Let s denote the source vertex. We say that a path $s \to j_1 \to \ldots \to j_k$ "passes through black vertices only" if the status of $s, j_1, \ldots, j_{k-1}$ is black. The end of the path, j_k, may or may not be black.

Consider the following three statements:

Q_0 : if vertex i is in the queue then status$(i) =$ grey.

Q_1 : $(\forall i, j \in V)(\text{ if } i \text{ is black and } j \text{ is not black then } c(i) \le c(j))$.

Q_2 : $(\forall i \in V)(c(i)$ is the minimum cost among all $s \to \ldots \to i$ paths that pass through black vertices only).

1. (a) Prove that Q_0 is a loop-invariant.
   (b) Prove that $Q_0 \& Q_1$ is a loop-invariant.
   (c) Prove that $Q_0 \& Q_1 \& Q_2$ is a loop-invariant.

2. Use these loop-invariants to prove the correctness of Dijkstra's algorithm. (Remark: for this we would really just need Q_2; but to prove that Q_2 holds throughout the execution of the algorithm, we need to rely on Q_1, and to prove that Q_1 always holds, we need Q_0. This results in the nested sequence of invariants above, a typical situation in proofs of correctness.)

3. (a) Prove that Q_1 alone is not a loop-invariant.

(b) Prove that $Q_0 \& Q_2$ is not a loop-invariant.

Explanation. You need to construct a weighted directed graph with nonnegative weights, a source, and an assignment of all the variables (parent links, status colors, current cost values, set of vertices in the queue) such that $Q_0 \& Q_2$ holds for your configuration and the **while** condition holds (the queue is not empty) but Q_2 will no longer hold after executing Dijkstra's **while** loop. Your graph should have very few vertices (4 vertices suffice).

4. For each statement, decide whether or not it is a loop-invariant for BFS: (a) "Vertex #2 is black." (b) "Vertex #2 is white." (c) "Vertex #2 cannot change from black to white." Reason your answers!