Análise e Técnicas de Algoritmos
Período 2003.1

Invariante de Laço

Invariante de Laço

Introdução

O conceito de invariante de laço é bastante importante e pode nos ajudar a analisar programas, verificar erros, e derivar programas a partir de especificações.

O invariante de laço pode ser definido como uma relação entre as variáveis de um algoritmo (programa) que é verdadeira nas seguintes condições:

- antes do início do laço;
- durante a manutenção do laço;
- na saída do laço.

Para ilustrar o conceito de invariante de laço, vamos considerar o seguinte algoritmo que computa a soma dos elementos de um array de inteiros.

int Soma(int A[])
$s \leftarrow 0$
$i \leftarrow 1$
while $i \leq A.length$ **do**
  $s \leftarrow s + A[i]$
  $i \leftarrow i + 1$
return s

É possível identificar invariantes de laço nesse exemplo?

Ainda sobre Laços

Em um laço é possível identificar dois elementos:

- **Guarda** é a expressão booleana que determina se o corpo do laço deve ser executado. O algoritmo executa as instruções do corpo do laço sempre que a guarda é avaliada verdadeira. Em nosso exemplo, a guarda é $i \leq A.length$.
- **Variante** é a expressão numérica que mede o quanto de execução ainda falta. A variante pode não estar explicitada no algoritmo (quase sempre isso acontece). Em geral, a variante é o número de iterações que ainda resta ocorrer. No nosso exemplo, a variante é $A.length - i + 1$.

Um laço está correto se as seguintes 5 condições são satisfeitas:

1. **Início:** O invariante é verdadeiro antes de entrar no laço.
2. **Invariância:** O corpo do laço preserva o invariante.
3. **Progresso:** O corpo do laço diminui a variante.

4. **Limitação:** Quando a variante atinge certo valor, a guarda se torna falsa.

5. **Saída:** A negação da guarda e o invariante descrevem o objetivo do laço.

Verificar no exemplo se as 5 condições são satisfeitas.

Um outro exemplo

O seguinte algoritmo computa x^n através de multiplicações sucessivas. Determine os invariantes de laço.

int Potencia(int x**, int** n**)**
$p \leftarrow 1$
$i \leftarrow 0$
while $i < n$ **do**
  $p \leftarrow p \times x$
  $i \leftarrow i + 1$
return p

Derivando Laços em um Algoritmo

A partir de invariantes, é possível derivar laços em um algoritmo. Para tanto, vamos considerar um padrão que relaciona as instruções de um laço:

PRE
$\triangleright$ Invariante é verdadeiro
while G **do**
  $\triangleright$ Invariante é verdadeiro
  C
  $\triangleright$ Invariante é verdadeiro
$\triangleright \neg G \wedge$ Invariante é verdadeiro
POS

G é a guarda do laço, PRE é o código que precede o laço, C corresponde ao corpo do laço e POS é o código que vem depois do laço.

Para ilustrar a derivação de laços, vamos considerar a definição da divisão de inteiros que relaciona as saídas q (quociente) e r (resto) às entradas n (número) e d (divisor). Por definição, sabemos que $r < d$ e $n = q \times d + r$.

De fato, a definição da divisão corresponde a pós-condição do laço (aquilo que é verdade após a execução do laço). Sabemos que na pós-condição, $\triangleright \neg G \wedge$ Invariante é verdadeiro. É fácil perceber que $\neg G = r < d$ e o invariante é $n = q \times d + r$. Logo, a guarda G é $r \geq d$. Substituindo no padrão definido acima temos:

PRE
$\triangleright\ n = q \times d + r$
while $r \geq d$ **do**
  $\triangleright\ n = q \times d + r$
  C
  $\triangleright\ n = q \times d + r$
$\triangleright \neg(r \geq d) \wedge\ n = q \times d + r$
POS

Devemos derivar ainda PRE e C, uma vez que POS pode ser (return q). A idéia básica é contar quantas vezes nós subtraímos o divisor d do numerador n. O quociente q é o contador e o resto r é o que sobra após a subtração. É justo considerar que inicialmente não fizemos nenhuma subtração. É importante ainda lembrar que o invariante deve ser válido antes do laço. Logo, se iniciamos $q \leftarrow 0$, indicando que não fizemos ainda nenhuma subtração, devemos fazer $r \leftarrow n$ para garantir o invariante.

Para definir o corpo do laço, devemos presevrar o inavariante. Logo, para cada subtração, o contador q é incrementado de 1 e o divisor subtraído do resto ($r \leftarrow r - d$).

O algoritmo derivado é, portanto,

$r \leftarrow n$
$q \leftarrow 0$
$\triangleright\ n = q \times d + r$
while $r \geq d$ **do**
  $\triangleright\ n = q \times d + r$
  $r \leftarrow r - d$
  $q \leftarrow q + 1$
  $\triangleright\ n = q \times d + r$
$\triangleright \neg(r \geq d) \wedge\ n = q \times d + r$
return q