Estruturas de Dados e Básicas I - DIM0119

Selan R. dos Santos

DIMAp – Departamento de Informática e Matemática Aplicada
Sala 231, ramal 231, `selan@dimap.ufrn.br`
UFRN

2018.1

Corretude de Algoritmos — Conteúdo

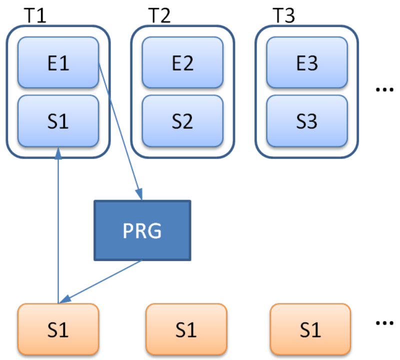
Motivação e Objetivos

- **Motivação**
  - ⋆ Ao desenvolver um algoritmo precisamos demonstrar de que dado uma entrada (pré-condição), ela gera uma saída esperada (pós-condição).
  - ⋆ Esta prova de corretude complementa a informação sobre complexidade estudada anteriormente, permitindo conhecermos a fundo (e comparar) um algoritmo.
- **Objetivos**
  - ⋆ Apresentar técnicas de verificação de demonstração de corretude de algoritmos.

Técnica de Testes

- Para verificar se um algoritmo está correto, a forma mais intuitiva seria produzir uma massa de testes e verificar se o resultado produzido pelo algoritmo corresponde ao resultado esperado.

1. Construir uma (vasta) massa de testes.
2. Executar algoritmo para cada entrada teste.
3. Compara o resultado produzido com o esperado.



Técnica de Testes

- Apesar de simples (vantagem), esta técnica possui desvantagens e limitações.
- Para alguns problemas computacionais, é trabalhoso produzir várias instâncias de um problema (i.e. o par *entrada* + *saída esperada*).
- É necessário elaborar casos de teste de tamanhos diferentes e com configurações distintas.
- Apenas garante corretude para a massa de testes.

Técnica de Indução

- Idealmente devemos garantir corretude para qualquer entrada que o algoritmo possa receber.
- Em algumas situações o algoritmo segue um mapeamento quase que direto de uma definição matemática.
- Nestes casos podemos faze uso da indução matemática.

Exemplo: n-ésima potência de um número.

- Calcular a n-ésima potência de um número x denotada por x^n.
- Pré-condição: $n \geqslant 0$.
- Pós-condição: retornar o valor correto de x^n.
- Definição matemática recursiva:

$$x^n = \begin{cases} 1 & \text{se } n = 0, \\ x \cdot x^{n-1} & \text{se } n > 0. \end{cases}$$

Técnica de Indução (cont.)

- Neste caso é possível produzir um mapeamento direto entre a definição matemática e o algoritmo.

Pseudo-código potenciação

```
função Pot(x: inteiro; n: inteiro): inteiro
    se n = 0 então
        retorna 1                      # Caso base da recursão.
    senão
        retorna x·Pot(x, n − 1)        # Chamada recursiva.
```

- Portanto, podemos utilizar indução matemática para provar corretude.

Técnica de Indução (cont.)

Prova da corretude de `Pot` por indução matemática sobre n

1. Queremos provar que `Pot` sempre retorna x^n para qualquer entrada.
2. ***Caso base.*** Se $n = 0$, então `Pot` retorna $1 = x^0 = x^n$.
3. ***Hipótese indutiva.*** Suponha `Pot(x,k)` retorna corretamente x^k, $\forall k, 0 < k \leqslant n$.
4. ***Passo indutivo.*** Precisamos provar que `Pot(x,k+1)` retorna x^{k+1}.
   Bom, se $k = 0$ então `Pot` está correto (caso base).
   Como $k + 1 > 0$, a função deve retornar $x \cdot \text{Pot}(x, (k+1) - 1)$, ou $x \cdot \text{Pot}(x, k)$.
   Visto que $\text{Pot}(x, k)$ retorna corretamente x^k (hipótese indutiva), podemos concluir que a função retorna $x \cdot x^k$, ou seja, a função retorna x^{k+1} □

Corretude para Algoritmos Não-Recursivos

- Para algoritmos recursivos é possível tentar estabelecer a corretude por meio de indução matemática.
- Para algoritmo não-recursivos precisamos determinar algumas informações e trabalhar um pouco mais na prova.
- Algoritmos não-recursivos se baseiam fortemente em laços.
- Para entendermos melhor como a prova de corretude se relaciona com laço precisamos definir o conceito de **invariante de laço** (IL).
- Outro conceito importante é determinar se um laço **termina**.

Exemplo: considere o seguinte algoritmo

```
// Pré-condição: assert limite ⩾ 0 & i = 0.
while( i != limite ) {
    i = i + 1;
}
// Pós-condição: assert i = limite.
```

- A pós-condição, $i = limite$, é verdadeira (satisfeita) após o laço?
- O laço termina?
- Se responder sim para as duas perguntas, o algoritmo é correto, segundo a especificação fornecida.
  1. Pré-condição garante $limite \geqslant i$.
  2. Para cada iteração do laço
     - $limite \geqslant i$ é verdadeiro no teste – se entrar no laço então $limite > i$.
     - i é incrementado em 1.
     - $limite$ não é modificado.
     - Portanto i se aproxima de $limite$ (mas $limite \geqslant i$ ainda é verdadeiro).
  3. Visto que existe apenas um número limitado de inteiros entre i e $limite$, podemos afirmar que eventualmente i será igual a $limite$.
  4. Execução sai do laço assim que $i = limite$ (mas $limite \geqslant i$ ainda é verdadeiro).

Compreendendo Laços por Indução

- A análise anterior é um argumento indutivo.
  - *Indução sobre o número de iterações.*
- Trabalhamos com indução computacional.
  - Mostramos que a conjectura é verdadeira para zero iterações.
  - Mostramos que ela permanece verdadeira depois de $n + 1$ iterações, assumindo que ela era verdadeira para n iterações.
- Precisamos provar duas coisas:
  1. Certa propriedade é preservada (denominada "corretude parcial"), a cada iteração do laço concluída.
     - *Invariante do laço é preservado por cada iteração, se a iteração é concluída.*
  2. O laço termina (denominada "terminação").
     - *A "função de decremento" ou "variante" é reduzida por cada iteração e não pode ser reduzida eternamente.*

Como escolher o invariante de laço (IL)?

Forma genérica do laço

```
{P}
while(b) S;
{Q}
```

- Encontre um invariante, IL, tal que
  1. $P \Rightarrow IL$ (verdadeiro inicialmente)
  2. $\{IL \wedge b\}\, S \,\{IL\}$ (verdadeiro se laço executa 1 vez)
  3. $(IL \wedge \neg b) \Rightarrow Q$ (estabelece pós-condição)
- É suficiente saber que se o laço terminar, Q será verdadeiro.
- Encontrar o invariante é a chave para compreender o laço.
- Argumentação indutiva é um método completo de prova.
  - *Se um laço satisfaz as pré/pós condições, então existe um invariante suficiente para prová-lo.*

Revisitando o exemplo anterior

Provando corretude parcial

```
// Pré-condição: assert limite ⩾ 0 & i = 0.
while( i != limite ) {
    i = i + 1;
}
// Pós-condição: assert i = limite.
```

- Um invariante adequado: $IL : limite \geqslant i$.
  1. $limite \geqslant 0 \;\&\; i = 0 \Rightarrow IL$ (verdadeiro inicialmente)
  2. $\{IL \wedge i \neq limite\}\, i := i + 1;\, \{IL\}$ (verdadeiro se laço executa 1 vez)
  3. $(IL \wedge \neg(i \neq limite)) \Rightarrow i = limite$ (estabelece pós-condição)

Corretude total em conjunto bem-ordenado

- **Corretude total = corretude parcial + terminação**.
- Ainda falta provar que o laço termina.
- Suponha que o laço sempre reduz o valor de certa variável.
- O laço termina se a variável for:
  - Número natural? Inteiro?
  - Número real não-negativo? Booleano?
  - Vetor de elementos?
- Podemos afirmar que o laço termina se os valores da variável são (um subconjunto de um) conjunto bem-ordenado.
- Um conjunto X é *bem-ordenado* se existe uma relação de ordem total $\leqslant$ sobre X e que para qualquer $Y \subset X, Y \neq \emptyset$, existe um elemento-mínimo em Y. Isto é, $\exists p \in Y$ tal que $p < q \; \forall q \in Y$.
  - Conjunto ordenado; e
  - Todo subconjunto não vazio possui elemento-mínimo.

Função de decremento

- A função de decremento mapeia um estado (variáveis do programa) para algum conjunto bem-ordenado.
  - *Dica: procure usar números naturais.*
- Esta afirmação facilita o raciocínio sobre término.
- Considere: `while(b) S;`
- Procuramos por $D(X)$, onde X é um estado, tal que
  1. Uma execução do laço reduz o valor da função:
     $\{LI \wedge b\}\, S\, \{D(X_{pos}) < D(X_{pre}) \wedge LI\}$
  2. Se o valor da função é mínimo, o laço termina:
     $(LI \wedge D(X) = \text{minVal}) \Rightarrow \neg b$

Revisitando o exemplo anterior

Provando término

```
// Pré-condição: assert limite ⩾ 0 & i = 0.
// Invariante do laço: limite ⩾ i.
// Função decremento: (limite − i).
while( i != limite ) {
    i = i + 1;
}
// Pós-condição: assert i = limite.
```

- Esta é uma boa função de decremento?
  1. O laço reduz o valor da função decremento?
     // assert $(i \neq limite)$; seja $d_{pre} = (limite_{pre} - i_{pre})$.
     $i = i + 1;$
     // assert $(limite_{pos} - i_{pos}) < d_{pre}$.
  2. Se a função atingir o valor mínimo, o laço finaliza?
     $(limite \geqslant i \wedge limite - i = 0) \Rightarrow (i = limite)$.

Escolhendo um invariante para laço

- Para laços você precisa identificar:
  - O invariante do laço; e
  - A função decremento.
- Depois use seu raciocínio para provar o objetivo de maneira apropriada.
- Se a prova não funcionar:
  - Talvez você tenha escolhido um invariante incorreto ou função de decremento inadequada. Escolha outro(s) e tente novamente!
  - Talvez o laço foi codificado de maneira errada; Corrija o código!
- Escolher invariante de laço de maneira automática é um tópico de pesquisa.

Exemplos diversos #1

```
int buscaBin( int A[], int x, int l, int r ) {
    int m = 0;
    if ( l > r )  return -1;
    else {
        m = ⌊(l + r)/2⌋;   // Calcular índice do meio.
        if ( x == A[ m ] )       return m;
        else if ( x < A[ m ] )  return buscaBin( A, x, l, m-1 );
        else                     return buscaBin( A, x, m+1, r );
    }
}
```

- Pré-condição: A é um vetor ordenado de inteiro de tamanho $n = r - l + 1$ e $n \geqslant 0$.
- Pós-condição: O algoritmo retorna o índice de x se $x \in A$ ou -1 se $x \notin A$.
- IL: Se $x \in A$, ele está em $A[l \dots r]$.
- Prova: Se $n = 0$ então $r < l$ logo a função retorna -1. No caso geral, supomos IL é V no início da função e independente da chamada recursiva, o vetor é sempre menor.

  Precisamos provar que o IL é preservado entre chamadas.
  - Se $x = A[m]$, ele ocorre em $A[l \dots r]$ e IL é V.
  - Se $x > A[m]$, então se $x \in A$ deve estar após m. Logo o IL é preservado se recursão aplicada sobre $A[m + 1 \dots r]$.
  - Se $x < A[m]$, então se $x \in A$ deve estar antes de m. Logo o IL é preservado se recursão aplicada sobre $A[l \dots m - 1]$.

  Em qualquer um dos casos anteriores, (1) IL é V e (2) chamadas recursivas são realizadas sobre vetores menores.

Exemplos diversos #2

Achar índice do elemento **máximo** em um vetor.

```
int max(int *arr, int sz) {
    auto k, idxMax( 0 );

    for(k = 1; k < sz; ++k) {
        // LOOP INVARIANTS HOLD HERE
        if (arr[k] > arr[idxMax])
            idxMax = k;
    }
    return idxMax;
}

PRE-CONDIÇÃO: arr é um vetor de inteiro de tamanho sz > 0
POS-CONDIÇÃO: para todo i: 0 <= i < sz --> arr[idxMax] >= arr[i]

INVARIANTE LAÇO: para todo i: 0 <= i < k --> arr[idxMax] >= arr[i]

FUNÇÃO DECREMENTO: (sz-k)
```


Exemplos diversos #3

Quadrado de um número.

```
int square(int n) {
    int i, sum;

    sum = 0;
    for (i = 0; i < n; i++) {
        //LOOP INVARIANTS HOLD HERE
        sum = sum + n;
    }
    return sum;
}

PRE-CONDIÇÃO: n >= 0
POS-CONDIÇÃO: sum = n * n

INVARIANTE LAÇO: Para qualquer 0 <= i < n, temos que 'sum = i * n'

FUNÇÃO DECREMENTO: (n-i)
```

Exemplos diversos #4

Fatorial de um número.

```
int factorial(int n) {
    int i = 1;
    int fat = 1;
    while ( i < n ) {
        i = i + 1;
        fat = fat * i;
    }
    return fat;
}

PRE-CONDIÇÃO: n >= 0
POS-CONDIÇÃO: fat = n!

INVARIANTE LAÇO: fat armazena i!
FUNÇÃO DECREMENTO: (n-i)
```


Exemplos diversos #5

Calcular a divisão como subtrações repetidas.

```
// Dividir n por d ou calcular n/d
std::pair<int,int> divisão(int n, int d) {
    int q = 0; // quociente
    int r = n; // resto
    while( r >= d ) { // ainda é possível subtrair
        q = q + 1;
        r = r - d; // subtrações sucessivas
    }
    return std::make_pair( q, r ); // retornar quociente e resto
}

PRE-CONDIÇÃO: n >= 0 e d > 0
POS-CONDIÇÃO: 'n' dividido por 'd' é igual a 'q' com resto 'r'

INVARIANTE LAÇO: (d * q) + r == n

FUNÇÃO DECREMENTO: (r-d) >= 0
```


Invariante de Laço — Resumo

- Pode ser definido como uma relação entre as variáveis de um algoritmo que é verdadeira nas seguintes condições:
  - Antes do início do laço;
  - Durante a manutenção do laço; e
  - Na saída do laço.
- Além do invariante do laço, temos os elementos:
  - Guarda: expressão booleana que indica se o corpo do laço deve ou não ser executado;
  - Variante: expressão numérica que mede o quanto de execução ainda falta. Também conhecido como função decremento.

Invariante de Laço

Exemplo: Calcular e retornar a soma dos elementos em um arranjo.

função Soma(A: **arranjo de inteiro**): **inteiro**
    var s: **inteiro** $\leftarrow 0$ # inicializando acumulador
    var i: **inteiro** $\leftarrow 0$ # variável auxiliar
    enquanto $i <$ **tam** A **faça**
        $s \leftarrow s + A[i]$ # acumulando soma
        $i \leftarrow i + 1$ # processar o próximo
    retorna s

- Pré-condição: $s = 0$ e **tam** A $\geqslant 0$.
- A guarda: $i <$ **tam** A.
- A variante: **tam** $A - i$ (ou função decremento).
- A invariante: Para todo $0 \leqslant i <$ **tam**A, temos $s = \sum^{i} A[i]$.
- Pós-condição: $s = \sum^{\text{tam A}} A[i]$ (soma de todos elementos de A).

Invariante de Laço — Resumo

- Uma laço está correto se as seguintes 5 condições são satisfeitas:
  1. **Início**: o invariante é verdadeiro antes de iniciar o laço.
  2. **Invariância**: o corpo do laço preserva o invariante.
  3. **Progresso**: o corpo do laço diminui o variante.
  4. **Limitação**: quando a variante atinge o valor certo, a guarda se torna falsa.
  5. **Saída**: a negação da guarda e o invariante descrevem o objetivo do laço.
- Formulamos a condição invariante do laço IL
  - IL deve permanece verdadeiro por todo o laço.
  - IL é relevante para a definição do problema: usamos IL no final do laço para provar a corretude do resultado.
- Feito isto, resta apenas provar que o algoritmo termina.

Prova de Corretude — Sumário

- Você recebe um problema P e um algoritmo A
  - Q formalmente define uma condição de corretude (pós-condição).
  - Assuma, por simplicidade, que A consiste de um único laço.

---

1. Formule um invariante IL.
2. **Inicialização** (para todas as entradas válidas)
   - prove que IL é verdadeiro logo antes da primeira execução da primeira instrução do laço
3. **Manutenção** (para todas as entradas válidas)
   - prove que **se** IL é verdadeiro logo antes da primeira instrução do laço, **então** IL também deve ser verdadeiro após a última instrução do laço.
4. **Terminação** (para todas as entradas válidas)
   - prove que o laço termina, ao satisfazer a certa condição de término b.
5. Prove que $IL \wedge \neg b \Rightarrow Q$, o que significa que A é **correto**$\square$

Exercício proposto #1

Filtrar os elementos positivos, preservando-os no início do vetor com sua ordem relativa mantida

```
void filter( std::vector<int> & V ) {
    auto iSlow( 0ul ); // Both indexes start at the first
    auto iFast( 0ul ); // (unprocessed) element.
    // Run through the array.
    for( /* Empty */ ; idxFast < V.size() ; ++iFast ) {
        // Check current unprocessed value.
        if ( V[ iFast ] > 0 ) // Do we need to include it?
            V[ iSlow++ ] = V[ iFast ]; // Copy positive integer to
                                       // the compacted region.
    }
    V.resize( iSlow ); // Adjust vector to the (possibly) new size.
}
PRE-CONDIÇÃO: V.size() >= 0
POS-CONDIÇÃO: All elements in V are > 0 and
              they preserve their original relative order.
INVARIANTE LAÇO: ?0 ⩽ i < n, ∀i ∈ [0; iSlow) ⇒ V[i] > 0
FUNÇÃO DECREMENTO: ?D=(V.size()-idxFast)
```


Referências

- D. Schmidt
  CIS301 Lecture Notes, **Cap. 4**.
  Kansas State University, 2001.
- A. Campos e C. Madeira
  Slides de Aula sobre Corretude. **Aula 03**
  IMD, UFRN, 2014.
- M. Ernst
  Slides de Aula sobre Invariantes de Laço. **Módulo CSE 331**
  University of Washington, 2010.