Disseny de programes iteratius: Derivació

Programació 2

Facultat d'Informàtica de Barcelona, UPC

Conrado Martínez

Primavera 2019

- Apunts basats en els d'en Ricard Gavaldà
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

Disseny inductiu

Disseny inductiu o derivació

Invertim el procés: de la "justificació" a l'algorisme

Donades Pre i Post, proposar:

- un invariant que les generalitzi les dues
- deduim les inicialitzacions que, amb la Pre, estableixin l'invariant
- un cos del bucle que mantingui l'invariant
- una condició del bucle que, negada, i junt amb l'invariant impliqui la Post

Exemple: Part entera de l'arrel quadrada

```
// Pre: x = X ≥ 0
   ????
// Post: a = ⌊√X⌋ (part entera per defecte
//       de l'arrel quadrada de X
```

Exemple: Part entera de l'arrel quadrada

```
// Pre: x = X ≥ 0
   ????
// Post: a = ⌊√X⌋ (part entera per defecte
//       de l'arrel quadrada de X
```

Post: $a^2 \leq X < (a+1)^2$ Invariant: $I = (x = X \geq 0) \wedge (a^2 \leq x < b^2)$

Exemple: Part entera de l'arrel quadrada

```
// Pre: x = X ≥ 0
   ????
// Post: a = ⌊√X⌋ (part entera per defecte
//       de l'arrel quadrada de X
```

Post: $a^2 \leq X < (a+1)^2$ Invariant: $I = (x = X \geq 0) \wedge (a^2 \leq x < b^2)$

```
int a = ?;
int b = ?;
while (B) {
    int c = (a+b)/2;
    if (?) a = c;
    else b = c;
}
```

Exemple: Part entera de l'arrel quadrada

```
int a = ?;
int b = ?;
while (B) {
    int c = (a+b)/2;
    if (?) a = c;
    else b = c;
}
```

- $I \wedge b \leq a+1 \implies a^2 \leq x < b^2 = (a+1)^2$, és a dir, $a = \lfloor \sqrt{x} \rfloor$ si $b \leq a+1$ (ja que $a \leq b$ sempre tindrem $b = a+1$); per tant la condició del bucle ha de ser la negació: `b > a+1`
- Com $0 \leq x = X < X^2$, fent `a = 0;` i `b = x+1;` establim l'invariant
- Si entrem al bucle tindrem $a < c < b$, i si $x < c^2$ llavors fent `b = c;` es torna a satisfer l'invariant; de manera similar, si $c^2 \leq x$ llavors fer `a = c;` reestablirà l'invariant.

Exemple: Part entera de l'arrel quadrada

```
int a = 0;
int b = x+1;
while (b > a+1) {
    int c = (a+b)/2;
    if (c * c <= x) a = c;
    else b = c;
}
```

El nostre algorisme és el conegut mètode de la bisecció per a trobar arrels de funcions continues

Nombre de parelles ordenades

Donat un vector v, comptar quantes parelles $(v[i], v[i+1])$ conté tals que $v[i] < v[i+1]$.

```
int parelles_ordenades(const vector<int>& v);
```

Nombre de parelles ordenades

Invariant:

$$I = \text{“}v.\texttt{size}() = 0 \lor$$
$$p = \text{nombre de parelles ordenades en } v[0..i-1] \land 1 \leq i \leq v.\texttt{size}()$$

- La variable i recorre el vector
- La variable p compta les parelles ordenades vistes fins al moment durant el recorregut

Nombre de parelles ordenades

1. Com establim l'invariant al principi?

```
int p = 0;
int i = 1; // v[0..0] no conté cap parella
```

Nombre de parelles ordenades

2. Quan acabem? I acabem satisfent la Post? La darrera parella que cal comprovar és $(v[n-2], v[n-1])$, amb $n = v.size()$. Si la nostra condició de sortida és $i = n$, hem provat totes les parelles $(v[j-1], v[j])$ amb $j < n$, inclós el cas $j = n-1$, que és la $(v[n-2], v[n-1])$ i ja hem acabat
   Podem posar de condició del bucle $i < v.\texttt{size}()$, que cobreix bé els casos $n = 0$ i $n = 1$, i que implica sortir quan $i = v.\texttt{size}()$

Nombre de parelles ordenades

3. Com avancem mantenim l'invariant?
   Volem avançar fent $i = i + 1$. Posem que ho fem com a darrera instrucció del cos del bucle.
   Per tant just abans d'incrementar s'hauria de complir que hem provat totes les parelles $(v[j-1], v[j])$ amb $j \leq i$.
   La que falta doncs és la parella amb $j = i$, que és $(v[i-1], v[i])$

```
 // I ∧ i < v.size()
if (v[i-1] < v[i]) p = p + 1;
i = i + 1;
// I
```

Nombre de parelles ordenades

```
int parelles_ordenades(const vector<int>& v) {
   int p = 0;
   for (int i = 1; i < v.size(); ++i) {
     // Inv: v.size() = 0 ó
     //      (p = nombre de parelles ordenades en v[0..i-1]
     //       i 1 ≤ i ≤ v.size())
     // Fita: 0 si v.size() = 0, v.size() - i altrament
     if (v[i-1] < v[i]) ++p;
   return p;
}
```

Ordenació

1. Ordenar un vector v: Deixar-lo de manera que
   "per a tot i, $0 \leq i \leq v.size() - 1$, $v[i] \leq v[i+1]$
2. Invariant:

   "$v[0..j]$ està ordenat" ...

   Fixem-nos que quan $j = v.size() - 1$ ja tenim tot el vector ordenat.

Ordenació

1. Ordenar un vector v: Deixar-lo de manera que "per a tot i, $0 \leq i \leq v.size() - 1$, $v[i] \leq v[i+1]$
2. Invariant:

   "$v[0..j]$ està ordenat" ...

   Fixem-nos que quan $j = v.size() - 1$ ja tenim tot el vector ordenat.

   "i tots els elements de $v[0..j]$ són més petits o iguals que els de $v[j+1..v.size()-1]$"

Ordenació

"$v[0..j]$ està ordenat i tots els elements de $v[0..j]$ són més petits o iguals que els de $v[j+1..v.size()-1]$"

Si incrementem j:

- $v[0..j]$ segueix ordenat! Per què?
- Però no és cert que "$v[0..j]$ és més petit que $v[j+1..v.size()-1]$"
- Només és cert si $v[j+1]$ era un element mínim de $v[j+1...v.size()-1]$
- Que hem de fer?
  - Buscar un valor mínim de $v[j+1...v.size()-1]$
  - Intercanviar-lo amb $v[j+1]$
  - Incrementant j es reestableix l'invaraint

Aquest és l'algorisme d'ordenació per selecció.

Exercici: Si no posem la segona part de l'invariant, deriveu la ordenació per inserció

Exemple: Prefix de suma màxima d'un vector

```
// Pre: v.size() > 0
// Post: el resultat és i tal que v[0] + ... + v[i] és màxima
//       i -1 ≤ i < v.size()
int psm(const vector<double>& v);
```

Exemple: Prefix de suma màxima d'un vector

```
// Post: el resultat és i tal que v[0] + ... + v[i] és màxima
//       i −1 ≤ i < v.size()
```

Què vol dir "és màxima"? Sigui $n = v.\texttt{size}()$ i definim $S_i = \sum_{k=0}^{i} v[k]$. Per conveni, $S_{-1} = 0$. Llavors estem dient que el resultat és el valor i, $-1 \leq i < n$, tal que

$$S_i = \max\{S_k \mid -1 \leq k < n\}$$

Exemple: Prefix de suma màxima d'un vector

Això suggereix que la nostra solució faci un bucle sobre j amb l'invariant:

`// Inv:` $S_i = \max\{S_k \mid -1 \leq k < j\} \wedge -1 \leq i < j < n$

Exemple: Prefix de suma màxima d'un vector

```
// Inv: −1 ≤ i < j < v.size(),
//      v[0] + ··· + v[i] ≥ v[0] + ··· + v[k] per a tot k ∈ [−1..j − 1],
//      sum = v[0] + ... + v[j − 1],
//      sumi = v[0] + ... + v[i]
```

Exemple: Prefix de suma màxima d'un vector

```
// Pre: v.size() > 0
int psm(const vector<double>& v) {
    int i = -1; int j = 0;
    double sum = 0; double sumi = 0;
    while (j < v.size()) {
        sum += v[j];
        if (sum > sumi) {
            sumi = sum;
            i = j;
        }
        ++j;
    }
    return i;
}
// Post: el resultat és i tal que v[0] + ... + v[i] és màxima
//       i -1 ≤ i < v.size()
```

Percentatge d'estudiants presentats (amb nota)

- Donat un conjunt d'estudiants (`Cjt_estudiants`) retornem el percentatge d'estudiants presentats (amb nota) del vector
- Especificació:

```
// Pre: C conté almenys un estudiant
// Post: el resultat es el percentatge de presentats de C
double presentats(const Cjt_estudiants& C);
```

Percentatge d'estudiants presentats (amb nota)

- Per saber el % de presentats calculem primer el nombre d'estudiants amb nota
- Tindrem una potscondició P' després del bucle: "$npres$ és el nombre d'estudiants presentants de C".
- Un cop tenim P', és immediat obtenir la postcondició de la funció Post
- Per obtenir P' recorrerem el conjunt C comptant els estudiants amb nota
- Invariant:

```
// Inv: 1 ≤ i ≤ C.mida() + 1
//      i npres = nombre d'estudiants amb nota entre
//      els primers i − 1 estudiants en ordre creixent
//      de DNI
```

Percentatge d'estudiants presentats (amb nota)

```
// Pre: C conté almenys un estudiant
double presentats(const Cjt_estudiants& C) {
    int npres = 0;
    for (int i = 1; i <= C.mida(); ++i) {
      // Inv: 1 ≤ i ≤ |C|,
      //       npres = nombre d'estudiants amb nota entre
      //       els i−1 primers
      if (C.consultar_iessim(i).te_nota())
         ++npres;
      // npres = nombre d'estudiants amb nota entre els
      // i primers
      ++i;
    }
    // P': npres és el nombre d'estudiants presentants de C
    return double(npres * 100)/C.mida();
}
// Post: el resultat és el percentatge d'estudiants
//       presentats de C
```

Arrodoniment de la nota

Donat un vector d'estudiants, modificar-lo arrodonint-ne les notes a la dècima més propera (es pot fer com a acció o com a funció).

```
// Pre: cert
// Post: vest té les notes dels estudiants arrodonides
// a la dècima més propera del seu valor inicial
void arrodonir_notes(vector<Estudiant>& vest);
```

Arrodoniment de la nota

Farem un recorregut pels elements del vector i suposem que disposem de la funció:

```
// Pre: cert
// Post: retorna el valor més proper a x amb un sol decimal
double arrodonirment(double x);
```

Arrodoniment de la nota

Invariant: igual que la postcondició però aplicada només a la part tractada del vector

```
// Inv: vest[0..i-1] té les notes dels estudiants arrodonides
//      a la dècima més propera del seu valor inicial,
//      0 <= i <= vest.size()
```

- Quan $i = v.\texttt{size}()$, $\texttt{Inv} \implies \texttt{Post}$
- Si cada iteració incrementa i, per mantenir l'invariant abans hem d'arrodonir $vest[i]$

Arrodoniment de la nota

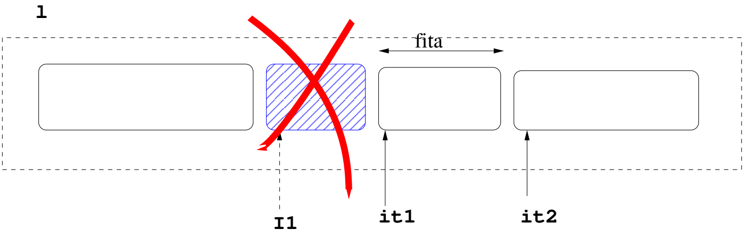
```
// Pre: cert
void arrodonir_notes(vector<Estudiant> &vest) {
  int n = vest.size();
  int i = 0;
  // Inv: ...
  while (i < n) {
    if (vest[i].te_nota()) {
      double aux = arrodoniment(vest[i].consultar_nota());
      vest[i].modificar_nota(aux);
    }
    ++i;
  }
}
// Post: vest té les notes dels estudiants arrodonides
// a la dècima més propera del seu valor inicial
```

Eliminació d'una subllista

```
  // Pre: it1 = I1 i it2 = I2 apuntem elements de la llista L i
  // it2 apunta a un element igual o posterior a l'element
  // apuntat per it1; l = L
  // Post: La llista l conté tots els elements d'L, excepte
  // els que hi havia entre it1 i el predecessor de it2, i.e.,
  // l = L[:it1) + L[it2:); '+' denota la concatenació de llistes
template <class T>
void elimina_subllista(list<T>& l,
    list<T>::iterator it1, list<T>::iterator it2);
```

Eliminació d'una subllista

Com a invariant proposem que la subllista entre $it1$ i el predecessor d'un iterador it ha sigut eliminada; quan $it = it2$ tindrem la postcondició:



Eliminació d'una subllista

Invariant "formal":

```
// Inv: it1 i it2 = I_2 apuntem elements de la llista L i
//      it2 apunta a un element igual o posterior a l'element
//      apuntat per it1, l = L[: I_1) + L[it1 :)
```

- L'invariant és compleix des del primer moment
- Si $it1 = it2$, llavors l'invariant implica la postcondició
- Si l'invariant és cert i $it1 \neq it2$, llavors eliminant l'element apuntat per $it1$ i avançant $it1$ reestablim l'invariant
- La funció de fita és la talla de $L[it1 : it2)$; la precondició (i l'invariant) garanteixen que és ≥ 0, i amb cada iteració disminuirà en una unitat

Eliminació d'una subllista

```
// Pre: it1 = I1 i it2 = I2 apuntem elements de la llista L i
// it2 apunta a un element igual posterior a l'element apuntat
// per it1; l = L
template <class T>
void elimina_subllista(list<T>& l,
        list<T>::iterator it1, list<T>::iterator it2) {

    while (it1 != it2)
      // Inv: it1 i it2 = I2 apuntem elements de la
      //      llista L i it2 apunta a un element
      //      igual o posterior a l'element apuntat
      //      per it1, l = L[:I1) + L[it1:)

      it1 = l.erase(it1);
}
// Post: La llista l conté tots els elements d'L, excepte
// els que hi havia entre it1 i el predecessor de it2, i.e.,
// l = L[:it1) + L[it2:); '+' denota la concatenació de llistes
```