Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2013

Departamento de Computación - FCEyN - UBA

Programación imperativa - clase 2

Teorema del Invariante

Asignación

Semántica de la asignación:
Sea e una expresión cuya evaluación no modifica el estado

$//$estado a
`v = e;`
$//$vale $v == e@a \wedge z_1 = z_1@a \wedge \cdots \wedge z_k = z_k@a$

donde $z_1, \ldots, z_k$ son todas las variables del programa en cuestión distintas a v que aparezcan en una cláusula *modifica* o *local*
Las otras variables se supone que no cambian así que no hace falta decir nada)

Si la expresión e es la invocación a una función que recibe parámetros por referencia, puede haber más cambios, pero al menos

$//$vale $v == e@a$

está en la poscondición de la asignación

Condicionales

`if (B) uno else dos;`

B tiene que ser una expresión booleana sin efectos secundarios (no tiene que modificar el estado). Se llama guarda.
uno y dos son bloques de instrucciones (entre llaves)
Si sabemos que:
$//$vale $P \wedge B$
`uno;`
$//$vale Q
Y si sabemos que:
$//$vale $P \wedge \neg B$
`dos;`
$//$vale Q
Entonces
$//$vale P
`if (B) uno else dos;`
$//$vale Q
donde P es la precondición del condicional y Q es su poscondición

Condicionales

En el caso :
$//$vale $P \wedge B$
`uno;`
$//$vale Q_1

En el caso :
$//$vale $P \wedge \neg B$
`dos;`
$//$vale Q_2

Entonces,
$//$vale P
`if (B) uno else dos;`
$//$vale $Q_1 \vee Q_2$

Ejemplo de demostración de condicional

problema $max(x, y : \text{Int}) = result : \text{Int}\{$
asegura $Q : (x > y \wedge result == x) \vee (x \leq y \wedge result == y)$
$\}$

```
int max(int x, int y) {
  int m = 0;
    //vale P_if : m == 0;
  if (x > y)
     m = x;
  else
     m = y;
    //vale Q_if : (x > y ∧ m == x) ∨ (x ≤ y ∧ m == y);
  return m;
    //vale Q;
}
```


Ciclos

```
while (B) cuerpo;
```

B: expresión booleana, sin efectos colaterales.
También se la llama guarda

cuerpo es un bloque de instrucciones (entre llaves)
Se repite mientras valga B , cero o más veces
Si es una cantidad finita, el programa termina
En tal caso, B tiene que llegar a ser falsa
y el estado final de esa ejecución será el estado final del ciclo.

Ejemplo de ciclo

problema $sumat(x : \text{Int}) = r : \text{Int}\{$
requiere $P : x \geq 0$
asegura $r == \sum[0..x]$ $\}$

En funcional

```
sumat :: Int -> Int
sumat 0 = 0
sumat n = n + (sumat (n-1))
```

En imperativo:

```
int sumat (int x) {
    int s = 0, i = 0;
    while (i < x) {
  // estado 1
        i = i + 1;
        s = s + i;
  // estado 2
    }
    return s;
}
```



```
int sumat (int x) {
    int s = 0, i = 0;
    while (i < x) {
  // estado 1
        i = i + 1;
        s = s + i;
  // estado 2
    }
    return s;
}
```

Estados para $x == 4$

$i@1$	$s@1$	$i@2$	$s@2$
0	0	1	1
1	1	2	3
2	3	3	6
3	6	4	10

Observar que en cada paso: $0 \leq i \leq x$
Cuando $i == x$, sale del ciclo $s == \sum[0..i]$
Estas dos valen en estado 1 y estado 2 (pero quizás no en el medio)

Otro ejemplo de ciclo

problema $fact(x : \text{Int}) = r : \text{Int}\{$
requiere $P : x \geq 0$
asegura $r == \prod[1..x]$
$\}$

En funcional

```
fact :: Int -> Int
fact 0 = 1
fact n = n * (fact (n-1))
```

En imperativo

```
int fact (int x) {
    int f = 1, i = 0;
    while (i < x) {
  // estado 1
        i = i + 1;
        f = f * i;
  // estado 2
    }
    return f;
}
```



```
int fact (int x) {
    int f = 1, i = 0;
    while (i < x) {
  // estado 1
        i = i + 1;
        f = f * i;
  // estado 2
    }
    return f;
}
```

Estados para $x == 4$

$i@1$	$f@1$	$i@2$	$f@2$
0	1	1	1
1	1	2	2
2	2	3	6
3	6	4	24

Observar que en cada paso: $0 \leq i \leq x$
Cuando $i == x$, sale del ciclo $f == \prod[1..i]$
Estas dos valen en estado 1 y estado 2 (pero quizás no en el medio)

Semántica de ciclos

Requiere cinco expresiones del lenguaje de especificación
una precondición P_C
una poscondición Q_C
un invariante I
una guarda B
una expresión variante v y una cota c.

```
...
  //vale P_C;
while (B) {    //invariante I;
  //variante v; cota c
  cuerpo
}
  //vale Q_C;
```


Invariante

Predicado cuya veracidad es preservada

- vale antes de entrar al ciclo (justo antes de evaluar la guarda)
- vale en cada iteración:
  justo después de entrar al cuerpo del ciclo, y
  justo después de ejecutar la última instrucción del cuerpo del ciclo,
  pero pude no valer en el medio del cuerpo.
- no hay forma algorítmica de encontrarlo
- conviene darlo al escribir el ciclo, porque encierra la idea del programador o diseñador

```
  //vale P_C;
while (B) {
  //invariante I;
  //variante v; cota c
  //vale I ∧ B;
  cuerpo
  //vale I;
}
  //vale Q_C;
```

Terminación y correctitud

```
  //vale P_C;
while (B) {
  //invariante I;
  //variante v; cota c
  //vale I ∧ B;
  cuerpo
  //vale I;
}
  //vale Q_C;
```

Un ciclo termina con respecto a la especificación del ciclo si no importa para qué valores de entrada que satisfagan P_C, el ciclo, después de una cantidad finita de pasos termina (es decir, se verifica $\neg B$).
Un ciclo es correcto (con respecto a la especificación del ciclo) si termina y al salir satisface Q_C

Terminación

¿cómo podemos garantizar que el ciclo termina?
Similares a conceptos que vimos para programación funcional
Para hablar de la cantidad de veces que se ejecuta un ciclo necesitamos:

Expresión variante (v)
Es una expresión del lenguaje de especificación, de tipo Int

- usa variables del programa
- debe decrecer en cada iteración

```
//estado a;
//vale B ∧ I;
cuerpo
//vale v < v@a;
```

Cota (c)
Es un valor entero (fijo, por ejemplo 0 o -8) que, si es alcanzado por la expresión variante, garantiza que la ejecución sale del ciclo
Formalmente: $(I \wedge v \leq c) \rightarrow \neg B$

Teorema de terminación

Si v es una expresión variante (con las propiedades de la p. **??**) e I es un invariante (con las propiedades de la p. **??**) de un ciclo y c es una cota tal que $(I \wedge v \leq c) \rightarrow \neg B$, entonces el ciclo termina.

Demostración.
Sea v_j el valor que toma v después de ejecutar el cuerpo del ciclo por j-ésima vez ($j \geq 0$)
Como v es estrictamente decreciente, $v_1 > v_2 > v_3 > \dots$
Como v es de tipo Int, $v_j \in Int$ para todo j
debe existir un k tal que $v_k \leq c$
como vale $(I \wedge v \leq c) \rightarrow \neg B$, tenemos que para este k vale $\neg B$ y por lo tanto sale del ciclo (después de iterar k veces)

$\square$

¿Qué pasaría si v fuese de tipo Float? ¿Funciona igual la demo?

Expresión variante y cota

Si v es una función variante con cota k,
$v' = v - k$ es una función variante con cota 0
Sin pérdida de generalidad, podemos suponer siempre una cota 0

Observaciones sobre terminación

```
int dec1(int x) {
while (x > 0)
    x = x - 1;
return x;
}
```

Supongamos que P_c garantiza $x \geq 0$.
El invariante debería ser $I : x \geq 0$
Expresión variante $v = x$, cota $c = 0$.
Guarda $B : x > 0$

$(I \wedge v \leq c) \rightarrow \neg B$?
Sí porque $x \leq 0 \rightarrow x \not> 0$.

Observaciones sobre terminación

```
int dec2(int x) {
while (x != 0)
    x = x - 1;
return x;
}
```

Supongamos que P_c garantiza $x \geq 0$.
El invariante debería ser $I : x \geq 0$
Expresión variante $v = x$, cota $c = 0$.
Guarda $B : not(x == 0)$

$(I \wedge v \leq c) \rightarrow \neg B$?
Sí
Notar que hace falta I ya que $x \leq 0 \not\rightarrow x == 0$
pero $(x \geq 0 \wedge x \leq 0) \rightarrow x == 0$

Teorema de Correctitud. Si un ciclo que termina satisface $P_C \rightarrow I$ y $(I \wedge \neg B) \rightarrow Q_C$ entonces el ciclo es correcto con respecto a su especificación.

Demostración. Queremos ver que el ciclo es correcto para su especificación, es decir, queremos ver que para variables que satisfagan P_C, cuando el ciclo termina se satisface Q_C.

```
  //estado 1
  //vale P_C;
while (B) {
  //vale I ∧ B;
  cuerpo
  //vale I;
}
  //estado 2
  //vale Q_C;
```

Supongamos que las variables en el estado 1 satisfacen P_C como $P_C \rightarrow I$, entonces en el estado 1 se satisface I.
Ejecutamos el ciclo (0 o más veces). Por definición de invariante (ver p. **??**), en cada iteración, el invariante se restablece. Por hipótesis, el ciclo termina y en el estado 2 vale $\neg B$ Además, en el estado 2 vale I. Como $(I \wedge \neg B) \rightarrow Q_C$ entonces en el estado 2 vale Q_C.□

Teorema del Invariante. Sea `while (B) { cuerpo }` un ciclo con guarda B, precondición P_C y poscondición Q_C. Sea I un predicado booleano, v una expresión variante, c una cota, y sean los estados A y B así

```
while (B) {
//estado A
cuerpo
//estado B}
```

Si valen

1. $P_C \rightarrow I$
2. $(I \wedge \neg B) \rightarrow Q_C$
3. el invariante se preserva en la ejecución del cuerpo, i.e. si $I \wedge B$ vale en el estado A entonces I vale en el estado B
4. v es decreciente, i.e. $v@A > v@B$
5. $(I \wedge v \leq c) \rightarrow \neg B$

entonces para cualquier valor de las variables del programa que haga verdadera P_C, el ciclo termina y hace verdadera Q_C, es decir, el ciclo es correcto para su especificación (P_C, Q_C).

Demostración. Inmediato del Teorema de Terminación (p. **??**) y el Teorema de Correctitud (p. **??**).□

Ejemplo de demostración de correctitud.
1. $P_C \rightarrow I$

```
int sumat (int x) {
    int s = 0, i = 0;
```
//vale $P_C : s == 0 \wedge i == 0$
```
    while (i < x) {
```
//invariante $I : 0 \leq i \leq x \wedge s == \sum[0..i]$
//variante $v : x - i$
```
        i = i + 1;
        s = s + i;
    }
```
//vale $Q_C : i == x \wedge s == \sum[0..x]$
```
    return s;
```
//vale $r == \sum[0..x]$
```
}
```

Supongamos que vale P_C y veamos que vale I

1. $i == 0$ implica $0 \leq i$.
2. $i == 0$ y $x == 0$ implica $i \leq x$.
3. $s == 0$ y $i == 0$ implica $\sum[0..i] == \sum[0..0] == 0 == s$.

Por lo tanto, $i == 0 \wedge s == 0 \rightarrow \ 0 \leq i \leq x \wedge s == \sum[0..i]$.□

Ejemplo de demostración de correctitud.
2. $(I \wedge \neg B) \rightarrow Q_C$

```
int sumat (int x) {
    int s = 0, i = 0;
```
//vale $P_C : s == 0 \wedge i == 0$
```
    while (i < x) {
```
//invariante $I : 0 \leq i \leq x \wedge s == \sum[0..i]$
//variante $v : x - i$
```
        i = i + 1;
        s = s + i;
    }
```
//vale $Q_C : i == x \wedge s == \sum[0..x]$
```
    return s;
```
//vale $r == \sum[0..x]$
```
}
```

Supongamos que vale $I \wedge \neg B$ y veamos que vale cada parte de Q_C.
Por I sabemos que $i \leq x$. Por $\neg B$ sabemos $i \geq x$. Entonces, $i == x$.
Por I sabemos $s == \sum[0..i]$, y recién mostramos que $i == x$, luego $s == \sum[0..x]$.□

Ejemplo de demostración. 3. El cuerpo preserva el invariante

```
int sumat (int x) {
    int s = 0, i = 0;
```
//vale $P_C : s == 0 \wedge i == 0$
```
    while (i < x) {
```
//invariante $I : 0 \leq i \leq x \wedge s == \sum[0..i]$
//variante $v : x - i$
```
        i = i + 1;
        s = s + i;
    }
```
//vale $Q_C : i == x \wedge s == \sum[0..x]$
```
    return s;
```
//vale $r == \sum[0..x]$
```
}
```

Hacemos la transformación de estados del cuerpo:

//estado 1
//vale $B \wedge I$
//implica $0 \leq i < x \wedge s == \sum[0..i]$ (justificación trivial)
```
        i = i + 1;
```
//estado 2
//vale $i == 1 + i@1 \wedge s == s@1$
```
        s = s + i;
```
//estado 3
//vale $i == i@2 \wedge s == s@2 + i@2$

Ejemplo de demostración. 3. El cuerpo preserva el invariante

//estado 1
//vale $B \wedge I$
//implica $0 \leq i < x \wedge s == \sum[0..i]$
```
        i = i + 1;
```
//estado 2
//vale $i == 1 + i@1 \wedge s == s@1$
```
        s = s + i;
```
//estado 3
//vale $i == i@2 \wedge s == s@2 + i@2$

Recordemos $I : 0 \leq i \leq x \wedge s == \sum[0..i]$, $B : i < x$
Sabemos que vale $(I \wedge B)@1$. Queremos ver que vale $I@3$.
$\boxed{\text{La transformación de estados indica } i@3 == i@2 == 1 + i@1.}$
Y por el estado 1 sabemos $i@1 \geq 0$. Luego $1 + i@1 \geq 0$. Por lo tanto, $i@3 \geq 0$.
Del estado 1 sabemos $i@1 < x@1$. Sumando 1 en ambos lados, $1 + i@1 < 1 + x@1$. Pero por la transformación de estados, $i@3 == 1 + i@1$, entonces $i@3 < 1 + x@1$ y esto es equivalente a $i@3 \leq x@1$.
La variable x no cambia porque es una variable de entrada que no aparece ni en *modifica* ni en *local*, entonces $x@3 == x@1$. Luego, $i@3 \leq x@3$.
Por la transfomación de estados $\boxed{s@3 == s@2 + i@2 == 1 + s@1 + i@1}$.
Como $s@1 == \sum[0..i@1]$, entonces usando que $i@1 + 1 == i@3$,
$s@3 == \sum[0..i@1] + 1 + i@1 == \sum[0..1 + i@1] == \sum[0..i@3]$.□

Ejemplo de demostración.
4. La expresión variante es decreciente.

Recordemos la transformación de estados

```
//estado 1
//vale B ∧ I
//implica 0 ≤ i < x ∧ s == ∑[0..i]
      i = i + 1;
//estado 2
//vale i == 1 + i@1 ∧ s == s@1
      s = s + i;
//estado 3
//vale i == i@2 ∧ s == s@2 + i@2
```

Debemos ver que la expresión variante $v : x - i$ es decreciente.
Es decir, debemos ver que $v@1 > v@3$.
Sabemos que x no cambia, luego $x@3 == x@1 == pre(x)$. Luego,
$v@1 == x@1 - i@1 == \mathrm{pre}(x) - i@1$ y $v@3 == x@3 - i@3 == \mathrm{pre}(x) - i@3$.
Por la transformación de estados, $i@3 = 1 + i@1$. Por lo tanto,
$v@1 == \mathrm{pre}(x) - i@1 > \mathrm{pre}(x) - (1 + i@1) == v@3$.$\square$

Ejemplo de demostración.
5. $(I \wedge v \leq c) \rightarrow \neg B$

```
int sumat (int x) {
    int s = 0, i = 0;
  //vale P_c : s == 0 ∧ i == 0
    while (i < x) {
  //invariante I : 0 ≤ i ≤ x ∧ s == ∑[0..i]
  //variante v : x − i
        i = i + 1;
        s = s + i;
    }
  //vale Q_c : i == x ∧ s == ∑[0..x]
    return s;
  //vale r == ∑[0..x]
}
```

Supongamos que vale $I \wedge v \leq c$. Debemos ver que vale $\neg B$.
Esta vez no hace falta usar I.
Como la cota es 0, $v \leq c$ es equivalente a $x - i \leq 0$,
que a su vez es equivalente a $x \leq i$;
y ésto es exactamente $\neg B$, ya que $B : i < x$.$\square$