Hola!

Algoritmos y Estructuras de Datos

Diego Bendersky, 11/9/2024

Qué vamos a ver hoy

- Ejercicios de WP y verificación de ciclos
- Introducción TADs

Faltan 2 semanas para el parcial!

Precondición más débil

3.1.1. Ejercicios de parcial

Ejercicio 7. Dado el siguiente condicional determinar la precondición más débil que permite hacer valer la poscondición (Q) propuesta. Se pide:

- Describir en palabras la WP esperada
- Derivarla formalmente a partir de los axiomas de precondición más débil. Para obtener el puntaje máximo deberá simplificarla lo más posible.

e) $Q \equiv \{(\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] \; mod \; 2 = 0)\}$

if $i \; mod \; 2 = 0$ **then**
| $s[i] = 2 * s[i]$
else
| $s[0] = 3;$
end

Sacar un caso del paratodo

$$(\forall j : \mathbb{Z})(0 \leq j < 5 \rightarrow_L P(j))$$

T	T	T	T	P(0)	P(1)	P(2)	P(3)	P(4)	T	T	T	T

$$\cdots \wedge T \wedge T \wedge P(0) \wedge P(1) \wedge P(2) \wedge P(3) \wedge P(3) \wedge P(4) \wedge T \wedge T \wedge \cdots$$

Sacar un caso del paratodo

$$(\forall j : \mathbb{Z})(0 \leq j < 5 \rightarrow_L P(j))$$

T	T	T	T	P(0)	P(1)	P(2)	P(3)	P(4)	T	T	T	T

Sacar un caso del paratodo

$$(\forall j : \mathbb{Z})(0 \leq j < 5 \rightarrow_L P(j))$$

T	T	T	T	P(0)		P(2)	P(3)	P(4)	T	T	T	T

$\wedge\, P(1)$

$$(\forall j : \mathbb{Z})(0 \leq j < 5 \wedge j \neq 1 \rightarrow_L P(j)) \wedge P(1)$$

Sacar un caso del paratodo

$$(\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L setAt(s, i, s[i] + 1)[j] > 0)$$

Cuanto vale $setAt(s, i, s[i] + 1)[j]$

si $i = j$ vale $s[i] + 1$
si $i \neq j$ vale $s[j]$

Sacamos el caso i del paratodo

$$(\forall j : \mathbb{Z})(0 \leq j < |s| \wedge j \neq i \rightarrow_L s[j] > 0) \wedge s[i] + 1 > 0$$

Correctitud de programas

pre

```
i = 0
sum = 0

while i < |s|
    sum = sum + s[i]

if sum > 3
    res = true
else
    res = false
```

post

Correctitud de programas

pre

S_1
```
i = 0
sum = 0
```

S_2
```
while i < |s|
    sum = sum + s[i]
```

S_3
```
if sum > 3
    res = true
else
    res = false
```

post

Correctitud de programas

pre

S_1
```
i = 0
sum = 0
```

Pc

S_2
```
while i < |s|
    sum = sum + s[i]
```

Qc

S_3
```
if sum > 3
    res = true
else
    res = false
```

post

Correctitud de programas

Hay que probar:

- pre => wp(S_1, Pc)
- Pc {S_2} Qc

(con teorema del invariante)

- Qc => wp(S_3, post)

pre

S_1
```
i = 0
sum = 0
```

Pc = pre + ejecución de S_1

S_2
```
while i < |s|
    sum = sum + s[i]
```

Qc = wp(S_3, post)

S_3
```
if sum > 3
    res = true
else
    res = false
```

post

Correctitud de ciclos

Dado el siguiente programa con su especificación

$P_c \equiv \{n > 0 \wedge n \ mod\ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

Contamos con el siguiente invariante, que sabemos que es incorrecto:

$I \equiv \{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{2(i-1)} j\}$

a) Señale qué axiomas del teorema del invariante no se cumplen. Justifique con palabras en forma precisa.
b) Escriba un invariante que resulte correcto.
c) Proponga una función variante y demuestre formalmente que es correcta.

Correctitud de ciclos

$$P_c \equiv \{n > 0 \wedge n \ mod \ 2 = 0 \wedge i = 1 \wedge res = 1\}$$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

Veamos la ejecución con n=6:

Iteración	i	res
0	1	1
1	2	1 * 1 * 6
2	3	1 * 1 * 6 * 2 * 5
3	4	1 * 1 * 6 * 2 * 5 * 3 * 4

= 6!

vemos que el ciclo funciona

Va multiplicando de los dos extremos y termina cuando i llega al medio

Correctitud de ciclos

Veamos qué pasa con el invariante:

$$I \equiv \{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{2(i-1)} j\}$$

i	res
1	$\prod_{j=1}^{0} j = 1$
2	$\prod_{j=1}^{2} j = 1 * 2$
3	$\prod_{j=1}^{4} j = 1 * 2 * 3 * 4$
4	$\prod_{j=1}^{6} j = 1 * 2 * 3 * 4 * 6$

= 6!

Tanto el código como el invariante funcionan pero lo calculan de forma diferente!

también funciona

Va multiplicando de a dos en dos y termina cuando i llega al medio

Correctitud de ciclos

Qué axiomas del invariante funcionan y cuáles no?

$P_c \equiv \{n > 0 \wedge n \ mod \ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

$I \equiv \{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{2(i-1)} j\}$

Pc => I (al entrar al ciclo)

$n > 0 \wedge nmod2 = 0 \wedge i = 1 \wedge res = 1$

reemplazo i=1 y res=1 en Inv y me queda:

$1 \leq 1 \leq n/2 + 1 \wedge 1 = \prod_{j=1}^{0} j$ ✓

Correctitud de ciclos

Qué axiomas del invariante funcionan y cuáles no?

$P_c \equiv \{n > 0 \wedge n \ mod\ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

$I \equiv \{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{2(i-1)} j\}$

I ∧ ¬B => Qc (al salir del ciclo)

si valen I ∧ ¬B entonces i = n/2+1. Reemplazamos res y nos queda:

$res = \prod_{j=1}^{n} j$ que es lo mismo que n! ✓

Correctitud de ciclos

Qué axiomas del invariante funcionan y cuáles no?

$P_c \equiv \{n > 0 \wedge n \ mod \ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

$I \equiv \{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{2(i-1)} j\}$

I $\wedge$ B {cuerpo del ciclo} I (durante del ciclo)

Tenemos que encontrar un caso que se cumpla el invariante y al ejecutar el código se deje de cumplir

Vemos con un ejemplo, usando la tabla que hicimos:

Iteración	i	res
0	1	1
1	2	1 * 1 * 6

$\prod_{j=1}^{0} j = 1$ cumple el invariante

$\prod_{j=1}^{2} j = 1 * 2$ no cumple el invariante

Correctitud de ciclos

$P_c \equiv \{n > 0 \wedge n \ mod \ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

b) Escriba un invariante que resulte correcto.

$$\boxed{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{i-1} j\,.\,(n+1-j)}$$

Iteración	i	res	invariante
0	1	1	$res = \prod_{j=1}^{0} j\,.\,(n+1-j) = 1$ ✓
1	2	1 * 1 * 6	$res = \prod_{j=1}^{1} j\,.\,(n+1-j) = 1.6$ ✓
2	3	1 * 1 * 6 * 2 * 5	$res = \prod_{j=1}^{2} j\,.\,(n+1-j) = 1.6.2.5$ ✓
3	4	1 * 1 * 6 * 2 * 5 * 3 * 4	$res = \prod_{j=1}^{3} j\,.\,(n+1-j) = 1.6.2.5.3.4$ ✓

Correctitud de ciclos

$P_c \equiv \{n > 0 \wedge n \ mod \ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

c) Proponga una función variante y demuestre formalmente que es correcta.

Necesitamos una función que:

- siempre se reduzca $\{I \wedge B \wedge v_0 = fv\}$ **S** $\{fv < v_0\}$
- si llega a cero se sale del ciclo $I \wedge fv \leq 0 \Rightarrow \neg B$,

Propuesta: $n/2 + 1 - i$

i va aumentando hasta llegar a n/2+1. Luego n/2+1-i se va achicando y vale cero cuando i = n/2+1

Correctitud de ciclos

- siempre se reduzca $\qquad fv = n/2 + 1 - i$

$$\{I \wedge B \wedge v_0 = fv\} \ \mathbf{S} \ \{fv < v_0\}$$

tenemos que probar que $I \wedge B \wedge fv = v_0 \rightarrow wp(cuerpo, fv < v_0)$

$wp(cuerpo, fv < v_0) =$

$$wp(res = res * i * (n + 1 - i); i = i + 1, n/2 + 1 - i < v_0)$$
$$wp(res = res * i * (n + 1 - i), wp(i = i + 1, n/2 + 1 - i < v_0))$$
$$wp(res = res * i * (n + 1 - i), n/2 + 1 - (i + 1) < v_0)$$
$$wp(res = res * i * (n + 1 - i), n/2 - i < v_0)$$
$$n/2 - i < v_0$$

Correctitud de ciclos

- siempre se reduzca $\qquad fv = n/2 + 1 - i$

$$\{I \wedge B \wedge v_0 = fv\}\ \mathbf{S}\ \{fv < v_0\}$$

tenemos que probar que $I \wedge B \wedge fv = v_0 \rightarrow\ n/2 - i < v_0$

para demostrar la implicación, asumimos que el precedente es verdadero
y queremos llegar a que el consecuente tiene que ser verdadero

si fv = v0, entonces $v_0 = n/2 + 1 - i$

reemplazamos a la derecha: $n/2 - i < n/2 + 1 - i$

lo cuál es siempre verdadero ✓

Correctitud de ciclos

- si llega a cero se sale del ciclo $fv = n/2 + 1 - i$

$$I \wedge fv \leq 0 \Rightarrow \neg B,$$

tenemos que probar que $I \wedge B \wedge n/2 + 1 - i \leq 0 \rightarrow i > n/2$

$$= I \wedge B \wedge n/2 + 1 \leq i \rightarrow i > n/2$$

$$= I \wedge B \wedge n/2 < i \rightarrow i > n/2 \quad ✓$$

TADs

Ejercicio 2. Especifique mediante TADs los siguientes elementos geométricos:

a) `Punto2D`, que representa un punto en el plano. Debe contener las siguientes operaciones:

   a) *nuevoPunto*: que crea un punto a partir de sus coordenadas x e y.

   b) *mover*: que mueve el punto una determinada distancia sobre los ejes x e y.

   c) *distancia*: que devuelve la distancia entre dos puntos.

   d) *distanciaAlOrigen*: que devuelve la distancia del punto $(0, 0)$.

b) `Rectangulo2D`, que representa un rectángulo en el plano. Debe contener las siguientes operaciones:

   a) *nuevoRectangulo*: que crea un rectángulo (decida usted cuáles deberían ser los parámetros).

   b) *mover*: que mueve el rectángulo una determinada distancia en los ejes x e y.

   c) *escalar*: que escala el rectángulo en un determinado factor. Al escalar un rectángulo un punto del mismo debe quedar fijo. En este caso el punto fijo puede ser el centro del rectángulo o uno de sus vértices.

   d) *estaContenido*: que dados dos rectángulos, indique si uno está contenido en el otro.

TADs

```
TAD Punto2D {

}
```

TADs

```
TAD Punto2D {

    proc nuevoPunto(in x: Z, in y: Z): Punto2D

    proc mover(inout p: Punto2D, in dx: Z, in dy: Z)

    proc distancia(in p: Punto2D, in p2: Punto2D): R

}
```

TADs

qué tipos pueden ir aca?

TAD Punto2D {

obs …

proc nuevoPunto(in x: Z, in y: Z): Punto2D

requiere …

asegura …

proc mover(inout p: Punto2D, in dx: Z, in dy: Z)

requiere …

asegura …

proc distancia(in p: Punto2D, in p2: Punto2D): R

requiere …

asegura …

}

TADs

Tipos para especificar

- int, float, char
- tupla, struct
- seq

Solo estos, no se puede componer!

Operación	Sintaxis
secuencia por extensión	$\langle\rangle$, $\langle x, y, z\rangle$
longitud	$\|s\|$, $length(s)$, $s.length$
pertenece	$i \in s$
indexación	$s[i]$
cabeza	$head(s)$
cola	$tail(s)$
concatenación	$concat(s_1, s_2)$, $s_1 ++ s_2$
subsecuencia	$subseq(s, i, j)$
cambiar elemento	$setAt(s, i, val)$

conj

Operación	Sintaxis
conjunto por extensión	$\{\}$, $\{x, y, z\}$
tamaño	$\|c\|$
pertenece	$i \in c$
union	$c_1 \cup c_2$
intersección	$c_1 \cap c_2$
diferencia	$c_1 - c_2$

dict

Operación	Sintaxis
diccionario por extensión	$\{\}$, $\{\text{"}juan\text{"} : 20, \text{"}diego\text{"} : 10\}$
tamaño	$\|d\|$, $length(d)$
pertenece (hay clave)	$k \in d$
valor	$d[k]$
setear valor	$setKey(d, k, v)$
eliminar valor	$delKey(d, k)$

funciones

$pertenece(e : T) : bool$

TADs

TAD Punto2D {

obs x: Z
obs y: Z

proc nuevoPunto(in x: Z, in y: Z): Punto2D

requiere {true}

asegura {res.x = x $\wedge$ res.y = y}

proc mover(inout p: Punto2D, in dx: Z, in dy: Z)

requiere {p = P_0}

asegura {p.x = P_0.x + dx $\wedge$ p.y = P_0.y + dy}

proc distancia(in p: Punto2D, in p2: Punto2D): R

requiere {true}

asegura {res = dist(p.x, p.y, p2.x, p2.y)}

aux dist(x1: Z, y1: Z, x2: Z, y2: Z): R

$$\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

}

TADs

Ejercicio 2. Especifique mediante TADs los siguientes elementos geométricos:

a) `Punto2D`, que representa un punto en el plano. Debe contener las siguientes operaciones:

a) *nuevoPunto*: que crea un punto a partir de sus coordenadas x e y.

b) *mover*: que mueve el punto una determinada distancia sobre los ejes x e y.

c) *distancia*: que devuelve la distancia entre dos puntos.

d) *distanciaAlOrigen*: que devuelve la distancia del punto $(0, 0)$.

b) `Rectangulo2D`, que representa un rectángulo en el plano. Debe contener las siguientes operaciones:

a) *nuevoRectangulo*: que crea un rectángulo (decida usted cuáles deberían ser los parámetros).

b) *mover*: que mueve el rectángulo una determinada distancia en los ejes x e y.

c) *escalar*: que escala el rectángulo en un determinado factor. Al escalar un rectángulo un punto del mismo debe quedar fijo. En este caso el punto fijo puede ser el centro del rectángulo o uno de sus vértices.

d) *estaContenido*: que dados dos rectángulos, indique si uno está contenido en el otro.

TADs

Ejercicio 4.

a) Especifique el TAD Diccionario$\langle K, V\rangle$ con las siguientes operaciones:

  a) *nuevoDiccionario*: que crea un diccionario vacío
  b) *definir*: que agrega un par clave-valor al diccionario
  c) *obtener*: que devuelve el valor asociado a una clave
  d) *esta*: que devuelve true si la clave está en el diccionario
  e) *borrar*: que elimina una clave del diccionario

TADs

Ejercicio 5. Especifique los TADs indicados a continuación pero utilizando los observadores propuestos:

a) $\boxed{\text{Diccionario}\langle K, V\rangle \text{ observado con conjunto (de tuplas)}}$

b) Conjunto$\langle T\rangle$ observado con funciones

c) Pila$\langle T\rangle$ observado con diccionarios

d) Punto observado con coordenadas polares

```
TAD Diccionario<K,V> {
        proc nuevoDicc(): Diccionario<K,V>

        proc definir(inout d: Diccionario<K,V>, in k: K, in v: V)
```

Hasta aca no cambia respecto del diccionario original!

```
        proc obtener(in d: Diccionario<K,V>, in k: K): V

        proc esta(in d: Diccionario<K,V>, in k: K): bool

}
```

TAD Diccionario<K,V> {

obs data: conj<tupla<K, V>>

proc nuevoDicc(): Diccionario<K,V>

requiere {true}

asegura {res.data =∅}

proc definir(inout d: Diccionario<K,V>, in k: K, in v: V)

requiere $\{d = D_0\}$

asegura $\{ < k, v > \in d.data\}$

asegura $\{(\forall t : tupla < K, V >)((t \in D_0.data \wedge t_0 \neq k) \rightarrow t \in d.data)\}$

asegura $\{estaPred(D_0.data, k) \rightarrow |d.data| = |D_0.data|\}$

asegura $\{\neg estaPred(D_0.data, k) \rightarrow |d.data| = |D_0.data| + 1\}$

proc obtener(in d: Diccionario<K,V>, in k: K): V

requiere $\{estaPred(d.data, k)\}$

asegura $\{(\exists t : tupla < K, V >)(estaPred(d.data, t_0) \wedge res = t_1)\}$

proc esta(in d: Diccionario<K,V>, in k: K): bool

requiere {true}

asegura $\{res = true \leftrightarrow estaPred(d.data, k)\}$

pred estaPred(in c: conj<tupla<K,V>>, k: K) =

$\{(\exists t : tupla < K, V >)(t \in c \wedge t_0 = k)\}$

}

Consulten!

Algoritmos y Estructuras de Datos

Diego Bendersky, 11/9/2024