DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Notas de Algoritmos y Estructuras de Datos I

Segundo Cuatrimestre 2022

26 de agosto de 2022 Algoritmos y Estructuras de Datos I

Integrante	LU	Correo electrónico
Gonzalez, Joan	51/22	jgquiroga@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

Parte I
Teoría

1. Corrección de Programas

1.1. SmallLang

Para poder hablar de corrección de programas, introducimos un lenguaje imperativo simplificado. Usamos SmallLang [1], que nos permite dar instrucciones y controlar el flujo de ejecución de dichas instrucciones con estructuras de control.

Instrucciones:

- Asignación – `x := E`
  Le asignamos a `x` el valor de la expresión `E`.
  Por ejemplo: `ejemplo := a*b + 2`
- Nada – `skip`.
  La instrucción indica que no debe ejecutarse nada.

De esta manera, **una instrucción es un programa**. Supongamos que `S1` y `S2` son programas, entonces `S1; S2` es un programa también. Además, si `B` es una expresión lógica, podemos usar estructuras de control.

Estructuras de control:

- Condicional – `if B then S1 else S2 endif` es un programa.
- Ciclo – `while B do S1 endwhile` es un programa.

Un ejemplo de programa puede ser:

```
x := 0
while (x > 0)
x := x+1;
x := x*2
endwhile
```

A medida que el programa se ejecuta, el valor de la variable `x` va cambiando. Al conjunto de todos los valores de las variables de un programa en un punto específico de ejecución, lo llamamos **estado**. De esta manera, la **ejecución** de un programa es una sucesión de estados.

Por ejemplo, veamos los estados del siguiente programa:

$\{a = A_0\}$
`c := a+2`
$\{a = A_0 \land c = A_0 + 2\}$
`result := c-1`
$\{a = A_0 \land c = A_0 + 2 \land result = (A_0 + 2) - 1 = A_0 + 1\}$

Como no sabemos el estado inicial de `a`, usamos la metavariable A_0 para indicar su valor inicial. Al proceso de ir evaluando consecutivamente los estados en función de las asignaciones que realiza un programa, lo llamamos *transformación* de estados. A través de la transformación de estados podemos determinar si un programa es correcto o no, dependiendo de su especificación.

Concretamente, decimos que un programa es **correcto** respecto de su especificación, si el programa comienza en un estado que cumpla la precondición, y termine su ejecución en un estado final que satisface la postcondición. Si el programa es correcto, entonces lo expresamos con la tripla de Hoare:

$$\{P\}S\{Q\} \tag{1}$$

Donde (P, Q) es la especificación, P es la precondición, Q es la postcondición y S es el programa correcto.

Para ejemplificar, tomemos la siguiente especificación:

```
proc incrementar (inout a: ℤ) {
    Pre {a = A_0}
    Post {a = A_0 + 1}
}
```

Y para ésta especificación, implementamos el siguiente programa:

```
proc incrementar (inout a: ℤ) {
    c := a+2;
    result := c-1;
    a := result
}
```

Pero por lo que hicimos previamente, vemos que el resultado de este programa es:

$$\{\texttt{a} = A_0 \land \texttt{c} = A_0 \texttt{ + 2} \land \texttt{result} = \texttt{(}A_0 \texttt{ + 2) - 1} = A_0 \texttt{ + 1}\}$$

Por lo que el programa es correcto, ya que cumple con todo lo pedido. En particular, el resultado cumple la postcondición. Ahora bien, teniendo un programa correcto respecto de su especificación, nos gustaría decir lo podemos escribir en forma de tripla de Hoare $\{P\}S\{Q\}$, es decir, queremos escribir algo del estilo:

```
proc incrementar (inout a: ℤ) {
    Pre {a = A_0}
    c := a+2;
    result := c-1;
    a := result
    Post {a = A_0 + 1}
}
```

Sin embargo, para poder hacerlo, debemos corroborar que sea una tripla válida. Decimos que una tripla de Hoare es válida si la precondición es la menos restrictiva para que se asegure la postcondición al final del programa. Ésta condición de *menos restrictiva* tiene una notación especial, decimos que es la *precondición más debil*, y la denotamos:

$$\text{Predicado } P \text{ más debil para que } S \text{ cumpla } Q \leftrightarrow \text{wp}(S, Q) \tag{2}$$

$$\text{La tripla } \{P\}S\{Q\} \text{ es válida} \leftrightarrow P \longrightarrow_L \text{wp}(S, Q) \tag{3}$$

Ahora bien, como sabemos que el programa implementado para la especificación es correcto, es decir que se cumple la postcondición si se cumple la precondición, y además la precondición no restringe la entrada, podemos decir que la tripla es válida. Pero, ¿Cómo haríamos con *cualquier* especificación y programa correspondiente?
Tomemos ahora como ejemplo la siguiente tripla:

```
proc mayorIgualaSiete (inout x: ℤ) {
    Pre {P}
    y := 2*x;
    x := y+1
    Post {Q : x ⩾ 7}
}
```

¿Qué precondición P hace que la tripla sea válida? Es decir, ¿Cuál es la precondición más débil de S para que se cumpla Q?

$$P = \text{wp}(S,Q)$$

Para poder deducir esto, necesitamos usar los axiomas de la precondición más debil:

1. wp(`x := E` , Q) $\equiv$ def(E) $\land_L$ Q^x_E
2. wp(`skip` , Q) $\equiv$ Q
3. wp($S1$ `;` $S2$, Q) $\equiv$ wp($S1$, wp($S2$,Q))

Acá introdujimos un poco de notación nueva:

- $\text{def}(E)$ – Son las condiciones necesarias para que E esté definida. Por ejemplo,
  - $\text{def}(x + y) = \text{def}(x) \land \text{def}(y)$
  - $\text{def}(\frac{x}{y}) = \text{def}(x) \land (\text{def}(y) \land_L y \neq 0)$
  - $\text{def}(\sqrt{x}) = \text{def}(x) \land_L x \geqslant 0$
  - $\text{def}(a[i] + 3) = (\text{def}(a) \land \text{def}(i) \land_L 0 \leqslant i < |a|)$

  Si además asumimos que $\text{def}(a) = \text{True}$ para cualquier a variable, entonces:
  - $\text{def}(x + y) = \text{True}$
  - $\text{def}(\frac{x}{y}) = y \neq 0$
  - $\text{def}(\sqrt{x}) = x \geqslant 0$
  - $\text{def}(a[i] + 3) = 0 \leqslant i < |a|$

- Q^x_E – Es el predicado que se obtiene reemplazando en Q todas las apariciones **libres** de x por E. Por ejemplo,

$$\begin{aligned} & Q \equiv 0 \leqslant i < j < n \land_L a[i] \leqslant x < a[j] \\ \rightarrow \quad & Q^i_{i+1} \equiv 0 \leqslant i + 1 < j < n \land_L a[i + 1] \leqslant x < a[j] \end{aligned} \tag{A}$$

$$\begin{aligned} & Q \equiv 0 \leqslant i < n \land_L (\forall j : \mathbb{Z})(a[j] = x) \\ \rightarrow \quad & Q^j_k \equiv 0 \leqslant i < n \land_L (\forall j : \mathbb{Z})(a[j] = x) \end{aligned} \tag{B}$$

Nótese que en la relación de equivalencia B, no podemos cambiar la variable j como lo indicaría el predicado, pues j es es una variable ligada y no libre.

Teniendo esto en cuenta, estamos listos para determinar en la tripla anterior cuál es la precondicion P más débil para que la tripla sea válida. Empecemos analizando el programa desde el fin de la ejecución, es decir, desde la postcondición, hacia el principio:

$\{Q : x \geqslant 7\}$
`x := y+1`
$\{\text{wp}(\texttt{x:=y+1}\ , Q)\}$

Pero, por lo que vimos previamente sabemos que:

$$\text{wp}(\texttt{x:=y+1}\ , Q) \equiv \text{def}(y + 1) \land_L y + 1 \geqslant 7 \equiv y \geqslant 6$$

Por ende,

$\{Q : x \geqslant 7\}$
`x := y+1`
$\{y \geqslant 6\}$
`y := 2*x;`
$\{P\}$

Y vemos que el valor de P debe ser, según lo visto:

$$P = \text{wp}(\texttt{x:=2*x}\ , R : y \geqslant 6) \equiv \text{def}(\texttt{2*x}\ ) \land_L R^y_{\texttt{2*x}} \equiv x \geqslant 3$$

Así, ya podemos reescribir las condiciones como una transformación de estados en orden, queda tal que:

$\{P : x \geqslant 3\}$
`y := 2*x;`
$\{R : y \geqslant 6\}$
`x := y+1`
$\{Q : x \geqslant 7\}$

Y además, la tripla de Hoare válida para `mayorIgualaSiete` queda:

```
proc mayorIgualaSiete (inout x: ℤ) {
    Pre {P : x ⩾ 3}
    y := 2*x;
    x := y+1
    Post {Q : x ⩾ 7}
}
```

1.2. Condicionales – Alternativas

Lo visto hasta ahora corresponde a la corrección de programas cuyo flujo de control es sólo secuencial, veamos ahora el caso de los condicionales. Tomemos como ejemplo la siguiente tripla:

```
proc alternativa (in a: ℤ, out c: ℤ) {
    Pre {P : a = A0}
    if a > 0
      c := a;
    else
      c := -a
    endif
    Post {Q : c = |a|}
}
```

¿Es correcta la postcondición? ¿Cómo podemos saberlo? Debemos corroborar que se cumpla en ambos casos.

- **Rama positiva** – $a > 0$

  Se cumple la condición $\rightarrow \{a = A_0 \land a > 0\} \equiv \{a = A_0 \land A_0 > 0\}$

  `c := a;` $\rightarrow \{a = A_0 \land c = A_0 \land A_0 > 0\}$

  $\Rightarrow \{c = |a|\}$

- **Rama negativa** – $a \leqslant 0$

  No se cumple la condición $\rightarrow \{a = A_0 \land \neg(a > 0)\} \equiv \{a = A_0 \land A_0 \leqslant 0\}$

  `c := a;` $\rightarrow \{a = A_0 \land c = -A_0 \land A_0 \leqslant 0\}$

  $\Rightarrow \{c = |a|\}$

Como en ambos casos vale que $\{c = |a|\}$ entonces la condición también vale luego del condicional y por ende la postcondición se cumple.

Pero ahora, ¿Cómo podemos saber si la tripla es válida? Es decir, teniendo un condicional, ¿Cómo podemos saber cuál es su precondición más debil?

1.3. Ciclos – Teorema del Invariante

Ahora es el turno de los ciclos, ¿Cómo podemos saber si son correctos o no? Recordemos que un ciclo en SmallLang se ve algo así:

```
while (B)
S;
endwhile
```

Donde `B` es la *guarda* y es una expresión lógica. Mientras `B` sea verdadera, se repite el cuerpo del ciclo, que en este caso ejecuta al programa `S` . A cada ejecución la llamamos *iteración*. Decimos que el ciclo es correcto si termina y además se cumple la invariante del ciclo.

En particular, para que un ciclo sea correcto debemos demostrar que:

1. Un predicado invariante I es verdadero antes de que comience el ciclo, tal que $P \rightarrow I$.
2. La tripla $\{I \land B\}S_i\{I\}$ se cumple para cada instrucción S_i del ciclo.
3. El predicado $P \land \neg B \rightarrow Q$ es verdadero. Es decir, que se cumpla la postcondición deseada.

4. El predicado $P \wedge B \rightarrow (t > 0)$ donde t es la cantidad de iteraciones. Es decir, queremos ver que t esté limitada inferiormente por 0, mientras el ciclo no haya terminado.

5. La tripla $\{I \wedge B\}$ `t1 = t;` S $\{t < t1\}$ valga para $1 \leqslant i \leqslant n$ de forma tal que esté garantizado que t decrezca por cada iteración del ciclo.

[1]

Las primeras tres demostraciones nos hablan de si se cumple la invariante I, y juntas forman el **Teorema del Invariante**. Si demostramos el Teorema del Invariante para un ciclo, entonces habremos demostrado que el ciclo es *parcialmente* correcto. Es decir, demostramos que si el ciclo termina, entonces cumplirá con Q. No obstante, las últimas dos demostraciones nos hablan de la seguridad de que el ciclo termine. Juntando todo, si demostramos que un ciclo cumple el Teorema del Invariante, y además demostramos que el ciclo siempre termina, entonces el ciclo será correcto.

Parte II

Bibliografía

[1] David Gries. The Science of Programming. Springer Science & Business Media, 2012.