Capítulo 10
El Comando Alternativo

Las notaciones de programación generalmente tienen un comando condicional, o si-instrucción, que permite que la ejecución de un subcomando dependa del estado actual de las variables del programa. Un ejemplo de un comando condicional, tomado de ALGOL 60 y Pascal, es

```
if x >= 0 then z:= x else z := -x
```

La ejecución de este comando almacena el valor absoluto de x en *z:* si x > 0 entonces la primera alternativa *z:=x* se ejecuta*;* de lo contrario, el segundo alternativa *z:=* -x se ejecuta. En nuestra notación de programación, este comando se puede escribir como:

```
(10.1) if  x  >=  0  ->  z:=  x
       |    x <=0    -> z:= -x
       end if
```

o, dado que es bastante corto y simple, en una línea como

```
if x >= 0 -> z:= x | x <= 0 -> z:=-x end if
```

El comando (10.1) contiene dos entidades de la forma *B -> S* (separadas por el símbolo "|") donde *B* es una expresión booleana y *S* un comando. *B -> S* se llama comando vigilado, porque *B* actúa como guardia en la puerta
->, asegurándose de que *S* se ejecute sólo en las condiciones adecuadas. Para ejecutar (10.1), debemos encontrar un guardian que me de verdadero y ejecutemos su programa.
Así, con x >0 ejecutar *z:=* x, con x <0 ejecuta *z* := —x, y con
x = 0 ejecuta *cualquiera* de las asignaciones (pero no ambas).

Esta breve introducción ha pasado por alto una serie de puntos importantes.

Seamos ahora más precisos al describir la sintaxis y la ejecución del comando alternativo.

La forma general del comando alternativo es

```
(10.2) if  B1  -> S1
       |   B2  -> S2
       .
       .
       .
       |   Bn -> Sn
       end if
```

donde $n >= 0$ y cada B_i -> S_i es un comando guardado. Cada S_i puede ser *cualquier* comando *saltar, cancelar,* composición secuencial, asignación, otro comando alternativo, etc.

Para propósitos de abreviatura, nos referimos al comando general (10.2) como IF, mientras que BB denota la disyunción

$$B_1 \vee B_2 \vee \ldots \vee B_n$$

El comando IF se puede ejecutar de la siguiente manera. Primero, si algún guardian B_i no es bien definido en el estado en que comienza la ejecución, puede ocurrir el aborto del programa.

Esto se debe a que no se asume nada sobre el orden de evaluación de los guardianes. En segundo lugar, al menos una guardia debe ser verdadera; de lo contrario la ejecución del IF aborta. Finalmente, si al menos una guardia es verdadera, entonces un comando guardado
Se elige B_i -> S_i con guardia verdadera B_i y se ejecuta S_i.

La definición de WP(IF, *Q*) ahora es bastante obvia.

El primera conjunción indica que las guardas deben estar bien definidas. La segunda conjunción indica que al menos un guardian es verdadero. El resto de conjunciones indican que la ejecución de cada comando S_i con una guarda verdadera B_i termina con Q true:

(I0.3a) Definición. $WP(IF, R) = dominio(BB)$ AND BB AND
B_1 -> $WP(S_1, Q)$ AND . . . AND B_n -> $WP(S_n, Q)$ ❑

Por lo general, asumimos que los guardianes son funciones totales, es decir, están bien definido en todos los estados. Esto nos permite simplificar la definición eliminando la pimera conjunción. Así, con la ayuda de cuantificadores reescribimos la definición en (10.3b) a continuación. De ahora

en adelante, usaremos (10.3b) como definición, pero asegúrese de que los guardias estén bien definidos en los estados en los que se ejecutará el comando alternativo!

(10.3b) Definición: WP(IF, *Q*) =

(Existe i: Z)(1 <= i <=n AND B_i *)* AND *(Para todo i: Z)(1 <= i <= n ->L* B_i *->WP(* S_i*, Q))*

Ejemplo 1.

Demostremos que, bajo todas las condiciones iniciales, la ejecución de (10.1) almacena el valor absoluto de x en *z.* Es decir, queremos demostrar que WP((10.1), *z = abs(x)) = Verdadero.* Tenemos:

WP ((10.1), z=*abs(x)) =*	|
(x >= 0 OR x <= 0) AND (	| BB AND
x > = 0 -> WP (z: = x, z = abs(x)) AND	| B_1 -> WP (S_1, Q) AND
x > = 0 -> WP (z: = x, z = abs(x))	| B_2 -> WP (S_2, Q)
)	|

Ejemplo 2.

Se supone que el siguiente comando es el cuerpo de un ciclo que cuenta el número de valores positivos *(p)* en el arreglo *b [0:m - 1].*

(10.4) If *b [i] > 0 -> p, i := p+1, i+1*
| *b [i] < 0 -> i := i + 1*
fi

Después de la ejecución de este comando, esperamos tener *i < m* y *p* igual a el número de valores en b [0: i-1] que son mayores que cero. Dejando *Q* ser la afirmación:

$$i <= m \text{ AND } p = \sum_{0}^{i-1} If\ b[i] > 0 \rightarrow 1 \mid b[i] \leq 0 \rightarrow 0\ fi$$

calculamos:

WP ((10.4), Q*) = (b [i] > 0* OR *b[i] < 0)* AND
(b [i] > 0 -> WP (p, i := p+1, i+1; Q) AND
(b [i] < 0 -> WP (p, i := p+1, i+1; Q)

Resolviendo la WP usando los axiomas de la teórica uba-aed1.

$$= b[i] <> 0 \text{ AND } i < m \text{ AND } p = \sum_{0}^{i-1} If\ b[i] > 0 \rightarrow 1 \mid b[i] \leq 0 \rightarrow 0\ fi$$

Por lo tanto, vemos que el arreglo *b* no debe contener el valor 0, y que la

definición de p tiene que ser el número de valores mayores que cero en *b [0: i -1].* Será verdadero después de la ejecución del comando alternativo si es verdadero antes la WP.

El lector puede sentir que hubo demasiado trabajo para probar lo que hizo en el ejemplo 2. Después de todo, el resultado se puede obtener de una forma intuitiva, y tal vez con bastante facilidad (aunque es probable que uno pase por alto el problema con cero elementos en el arreglo *b).* En este punto, es importante practicar tales manipulaciones formales. Esto dará una mejor comprensión de la teoría y una mejor comprensión del comando alternativo en sí.

Además, el tipo de manipulaciones realizadas en el ejemplo 2 ciertamente será necesario en el desarrollo de algunos programas, y la facilidad necesaria para esto sólo puede venir a través de la práctica.
Incluso el acto de realizar algunos ejercicios comenzará a cambiar la forma en que "naturalmente" piensa sobre los programas y así lo que llamas tu intuición sobre la programación.

Posteriormente, al atacar un problema similar al trabajado antes, puede que no sea necesario ser tan formal, pero la formalidad se dará a su alcance cuando lo necesite en los problemas más difíciles.

Algunos comentarios sobre el comando alternativo

El comando alternativo difiere de la sentencia if convencional en muchos aspectos.

Ahora discutimos las razones de estas diferencias.

Primero, el comando alternativo permite cualquier número de alternativas, no sólo dos. Por lo tanto, también sirve como una "instrucción de caso" (Pascal) o "SELECCIONAR declaración" (PL/I).

No hay necesidad de tener dos notaciones diferentes, una para dos alternativas y una para más.

Una notación para cada concepto -es un principio bien conocido y razonable-

No hay valores predeterminados: cada comando alternativo debe ir precedido de una guarda que describe las condiciones bajo las cuales puede ejecutarse.

Por ejemplo, el comando para asignar a x el absoluto de x debe escribirse con dos comandos guardados:

```
If  x  >=  0  -> skip
|    x <=0    -> x:= -x
```

```
end if
```

Su contraparte en ALGOL, **`If x <0 then z:= -x`**, tiene por defecto que si la ejecución de x >= 0 es equivalente a la ejecución de *skip* . Aunque un programa puede ser un poco más largo debido a la falta de un valor predeterminado, hay ventajas.

La aparición explícita de cada guarda ayuda al lector; cada alternativa se da con todo detalle, dejando menos posibilidades de pasar por alto algo.

Más importante aún, la falta de un valor predeterminado ayuda durante el desarrollo del programa.

Al derivar un posible comando alternativo, el programador es *obligado* a derivar las condiciones bajo las cuales su ejecución se llevará a cabo satisfactoriamente y, además, se *ve obligado* a seguir derivando alternativas hasta que al menos uno sea verdadero en cada estado inicial posible. Este punto quedará más claro en la Parte III.

La ausencia de valores predeterminados, de forma razonable, da la posibilidad del no determinismo.

Suponga que x = 0 cuando se ejecuta el comando (10.1). Entonces, dado que ambas guardas x <= 0 y x >= 0 son verdaderas, *cualquiera* se puede ejecutar (pero solo uno de ellos). La elección depende total mente del ejecutor, por ejemplo, podría ser una elección aleatoria, o en días con fechas impares podría ser el primero y en días con fechas pares podría ser el segundo, o podría elegirse para minimizar el tiempo de ejecución de un programa.
El punto es que, dado que la ejecución de cualquiera de los dos conduce a un resultado correcto, el programador no debería tener que preocuparse por cuál se ejecuta.

Él es libre de derivar tantos comandos alternativos y guardias como sea posible, sin tener en cuenta la superposición.

Por supuesto, para fines de eficiencia, el programador podría fortalecer los guardianes para extirpar el no determinismo. Por ejemplo, cambiando el segunda guardián en (10.1) de x <= 0 a x < 0 ayudaría si la evaluación de en el caso x = 0 entonces podría ejecutarse z := x.

Finalmente, la falta de incumplimiento permite la posibilidad de simetría (ver (10.1)), lo cual agrada —si no es que es necesario— a alguien con conocimientos matemáticos.

Un teorema sobre el comando alternativo

Muy a menudo, no estamos interesados en la precondición más débil de un comando alternativo, pero sí para determinar si una condición previa conocida lo implica. Por ejemplo, si el comando alternativo aparece en un programa, es posible que ya sepamos que su precondición es la pos condición del comando anterior, y realmente no necesitamos calcular la condición previa más débil. En tales casos, el siguiente teorema es útil.

(10.5) Teorema. Considere el comando IF. Supongamos que un predicado *P* satisface

(1) *P* -> BB
(2) *P* AND *Bi* -> *WP (Si , Q),* para todo *i , 1 <= I <= n*

Entonces (y solo entonces) *Q -> WP (If, Q).* ❑

Prueba. Demostración dada en la teórica uba-aed1

Ejemplo 3.

Supongamos que se está realizando una búsqueda binaria para un valor x que está en el arreglo *b [0: n* -1]. Estamos en la etapa en la que los siguientes predicado *P* es verdadero:

P: ordenado (b [0: n - 1]) AND 0 <= *i < k < j <n* AND *x in b [i: j].*

Es decir, la búsqueda se ha reducido a la sección de matriz *b [i: j],* y *k* es un índice de esta sección. Queremos demostrar que

(10.6) {P} **If** *b[k] <= x -> i: = k | b[k] > x -> j: = k* **fi** {x in *b [i: j]}*

sostiene el primer supuesto *P* -> BB del teorema (10.5). Se cumple, porque el la conjunción de las guardas en (10.6) es equivalente a *Verdadero.* La segunda la suposición se mantiene, porque

P AND b [k] <= x -> *x in b [k:j]* = *WP (i:= k, x in b[i:j]),* y
P AND b [k] <= x -> *x in b [i:k]* = *WP (j:= k, x in b[i:j]).*

Las dos implicaciones se derivan del hecho de que *P* indica que el arreglo está ordenado y que x está en *b [i: j]* y de la segunda conjunción de los antecedentes. Por tanto, el teorema nos permite concluir que (10.6) es verdadero.

Ejercicios para el Capítulo 10 SOLUCIONES ABAJO (Ver enunciado en el libro)

1. Soluciones a mano adjuntas

2. Ver el enunciado de los problemas en el libro de The science of programming-Gries (pag.137)

EJERCICIO CAP 10: (Pag 137. Libro Gries)

① wp (IF Fi, Q) para algun predicado Q.

PARTIENDO DE LA DEFINICIÓN, EL IF GENERAL NO TIENE comandos guardados POR LO QUE NO TIENE SENTIDO HABLAR DE ESA ESTRUCTURA Y más aún de la WP.

② Probar que el comando IF SATISFACE:

a) LEY del milagro excluido $wp(S,F) = F$

b) Distributiva de la conjunción $wp(S,Q) \wedge wp(S,R) \rightarrow wp(S, Q \wedge R)$

SOLUCIÓN:

DADO un

IF $B_1 \rightarrow S_1$
$B_2 \rightarrow S_2$
$\vdots$
$B_i \rightarrow S_i$
$\vdots$
$B_n \rightarrow S_n$

Fi

a)

POR DEFINICIÓN alguna es verdadera, ~~ASUMIENDO QUE B_i ES VERDAD~~

y $(\forall i:\mathbb{Z})(\underbrace{1 \le i \le n}_{V} \rightarrow \underbrace{wp(S_i, Q)}_{F}) \equiv F$

F

PARA los ENTEROS FUERA del RANGO $1 \le i \le n$ no hay problema pues esa EXPRESIÓN SEGUIRÍA SIENDO FALSA por los ENTEROS que ESTÁN ADENTRO.

b)

Por DEFINICIÓN alguna es verdadera $(B_1 \vee B_2 \vee \dots \vee B_n)$

y

$(\forall i:\mathbb{Z})(1 \le i \le n \rightarrow wp(S_i, Q)) \wedge (\forall i:\mathbb{Z})(1 \le i \le n \rightarrow wp(S_i, R))$

VALE PARA un i PARTICULAR "a"

$(\underbrace{1 \le a \le n}_{P} \rightarrow wp(S_a, Q)) \wedge (\underbrace{1 \le a \le n}_{P} \rightarrow wp(S_a, R))$

$(\neg P \vee wp(S_a, Q)) \wedge (\neg P \vee wp(S_a, R))$ (Distributiva)

$((\neg P \vee wp(S_a, Q)) \wedge \neg P) \vee (\neg P \vee wp(S_a, Q)) \wedge wp(S_a, R))$ (Absorción)

$\neg P \vee (\neg P \wedge wp(S_a, R)) \vee (wp(S_a, Q) \wedge wp(S_a, R))$ (Absorción)

$\neg P \vee (wp(S_a, Q) \wedge wp(S_a, R))$

$P \rightarrow wp(S_a, Q) \wedge wp(S_a, R)$ ▨

y como vale para todo a en Rango queda demostrado.

③ S3: IF $w \leq r \rightarrow r, q := r-w, q+1$
| $w > r \rightarrow$ skip
Fi

Calcular $wp(S_3, \overbrace{qw+r = x \wedge r \geq 0}^{Q}) \equiv$

$(w \leq r \vee w > r)$ ← True porque $P \wedge \neg P = T$ $\wedge$ (

$w \leq r \rightarrow wp(r, q = r-w, q+1; Q) \wedge$

$w > r \rightarrow \underbrace{wp(skip, Q)}_{Q}$ ← Por axioma wp

)

FALTARÍA CALCULAR

$wp(r, Q = r-w, q+1;\ qw+r = x \wedge r \geq 0) \equiv$

$r \geq w \wedge (q+1)w + r - w = x \equiv r \geq w \wedge wq + r + w - w = x \equiv$

$\equiv r \geq w \wedge wq + r = x$

④ S4: IF $a > b \rightarrow a := a-b$
| $b > a \rightarrow b := b-a$
Fi

Calcular:

$wp(S_4, a>0 \wedge b>0) \equiv R$

$R \equiv (a>b \vee b>a) \wedge ($

$a>b \rightarrow wp(a := a-b, a>0 \wedge b>0) \wedge$

$a<b \rightarrow wp(b := b-a, a>0 \wedge b>0)$

)

$\underline{wp(a := a-b, a>0 \wedge b>0)} \equiv a>b \wedge b>0$

$\underline{wp(b := b-a, a>0 \wedge b>0)} \equiv a>0 \wedge b>a$

Entonces $R \equiv (a>b \vee b>a) \wedge ((a>b \rightarrow a>b \wedge b>0) \wedge$
$(a<b \rightarrow b>a \wedge a>0))$

$\equiv (a>b \vee b>a) \wedge ((a \leq b \vee (a>b \wedge b>0)) \wedge (a \geq$

Recordando la propiedad: $P \rightarrow P \wedge Q \equiv P \rightarrow Q$
$\neg P \vee (P \wedge Q)$
$\neg P \vee Q$

$R \equiv (\underbrace{a>b}_{P} \vee \underbrace{b>a}_{Q}) \wedge (\underbrace{a>b}_{P} \to \underbrace{b>0}_{R} \wedge \underbrace{a<b}_{Q} \to \underbrace{a>0}_{S})$ {Distributiva}

Simplificando:

$(P \vee Q) \wedge ((P \to R) \wedge (Q \to S))$ # Y AES UNA EXPRESIÓN QUE TIENE MUCHAS UTILIDADES.

⑤ CALCULATE $wp(S_5, x \leq y) \equiv R$

S_5: IF $x > y \to x, y := y, x$
| $x \leq y \to$ SKIP
FI

$R \equiv (x > y \vee x \leq y) \wedge$ (TRUE)

$x > y \to wp(x, y := y, x ; x \leq y) \wedge$
$x \leq y \to wp(\text{Skip} ; x \leq y)$
)

$wp(x, y := y, x ; x \leq y) \equiv y \leq x$

$R \equiv (x > y \to y \leq x) \wedge (x \leq y \to x \leq y)$ → TRUE

$R \equiv x \leq y \vee y \leq x$

NOTA: Podría concluir que $x = y$.

⑥ $f[0:n] \wedge g[0:m]$ son listas ordenadas alfabéticamente de nombres de personas.
~~Existe un x que está~~ Se sabe que almenos un nombre está en ambas listas. Sea x el primero dicho nombre.

S_6: IF $f[i] < g[j] \to i := i+1$
| $f[i] = g[j] \to$ SKIP
| $f[i] > g[j] \to j := j+1$
FI

$wp(S_6; Q) \equiv (f[i] < g[j] \vee f[i] = g[j] \vee f[i] > g[j]) \wedge ($

$f[i] < g[j] \to wp(i = i+1, Q) \equiv \text{orde} \wedge f[i+1] \leq x \wedge g[j] \leq x$
$f[i] = g[j] \to wp(\text{skip}, Q) \equiv \text{orde} \wedge f[i] \leq x \wedge g[j] \leq x$
$f[i] > g[j] \to wp(j := j+1, Q) \equiv \text{orde} \wedge f[i] \leq x \wedge g[j+1] \leq x$

{Q: Ordenado($f[0:n]$) $\wedge$ Ordenado($g[0:m]$) $\wedge$ orde $\wedge$ $f[i] \leq x \wedge g[j] \leq x$}

y para el rango debo pedir que $f[i] \wedge g[j]$ estén bien definidas $0 \leq i < |b| \wedge 0 \leq j < |b| \wedge wp(S_6, Q)$.

⑦ DUDA!

$\{ y>0 \wedge z + yx = ab \}$

IF impar(x) → z, x := z+y, x-1
| par(x) → skip
Fi

$Q: \{ y \geq 0 \wedge z + xy = ab \}$

y, x := 2y, x/2 } NO MODIFICO EL ESTADO, ES una línea que solo ocupa memoria.

$\{ y \geq 0 \wedge z + yx = ab \}$

① P → BB ? Sí pues $yx = ab - z$ y puede ser par o impar

②
- $y \geq 0 \wedge z + xy = ab \wedge impar(x) \rightarrow wp(z, x = z+y, x-1 ; Q)$

$y \geq 0 \wedge z + xy = ab \wedge impar(x) \rightarrow y \geq 0 \wedge z + y + (x-1)y = ab$

$y \geq 0 \wedge z + xy = ab$ ▨

la inferencia es obvia.

- $y > 0 \wedge z + yx = ab \wedge Par(x) \rightarrow y \geq 0 \wedge z + xy = ab$

$x \bmod 2 = 0$

$z + yx \equiv ab \pmod 2$
$z \equiv ab \pmod 2$
$0 \equiv ab - z \pmod 2$ → y puede ser par o impar.

EN ▨ ~~$z + xy = ab$ (cambiamos)~~
Como $ab - z$ es impar y x impar, 'y' solo puede ser impar

NOTA: LAS GUARDAS ESTAN DE MAS y podríamos optimizar el código.