Corrección de Programas Imperativos

Algoritmos y Estructura de Datos I
UNRC
Pablo Castro

Especificando Programas Imperativos

Los programas imperativos son bastante diferentes a los funcionales ya que tienen la noción de **estado:**

- Un programa imperativo consiste de variables de programación + sentencias de programación
- Un estado es un momento particular durante la ejecución del programa que puede ser caracterizado por el valor de las variables en ese momento.
- Las instrucciones de los programas imperativos permiten realizar cambios de estados.

Estados

Veamos un pequeño ejemplo con Dafny:

```
x := 1
```

$x \mapsto 1$

```
y := x + 1
```

$x \mapsto 1, y \mapsto 2$

```
x := 5
```

$x \mapsto 5, y \mapsto 2$

```
z := x + 5
```

Esta es una asignación

Estado

Sentencia que cambia el estado del programa

Una sentencia se dice **“pura”** si no cambia el estado del programa. En otro caso se dice que tiene **“efectos colaterales”.**

Estados y Predicados

Podemos pensar los predicados como conjuntos de estados:

$x \geq 0 \wedge y \geq 0$

$\{(x \mapsto 1, y \mapsto 2), (x \mapsto 0, y \mapsto 0), (x \mapsto 1, y \mapsto 1), \ldots\}$

$x + y \geq 10$

$\{(x \mapsto 5, y \mapsto 5), (x \mapsto 3, y \mapsto 8), (x \mapsto 1, y \mapsto 10), \ldots\}$

$True$

Todos los estados posibles

$False$

Conjunto vacío de estados.

Predicados más fuertes y más débiles

Decimos que un predicado P **es más fuerte** que el predicado Q si el conjunto de estados que define P está incluido en el conjunto de estados que define Q

$x \geq 10$ denota el conjunto $\{x \mapsto 10, x \mapsto 11, x \mapsto 12, \ldots\}$

$x > 10$ denota el conjunto $\{x \mapsto 11, x \mapsto 12, \ldots\}$

Tenemos que $x > 10 \Rightarrow x \geq 10$, se dice que $x > 10$ **es más fuerte** que $x \geq 10$.

Veamos que: $\{x \mapsto 10, x \mapsto 11, x \mapsto 12, \ldots\} \supseteq \{x \mapsto 11, x \mapsto 12, \ldots\}$

Pre y Post Condiciones

Las pre y postcondiciones permiten verificar programas imperativos:

- Las pre/post-condiciones son predicados lógicos (proposicionales o de primer orden).
- Los programas pueden escritos en cualquier lenguaje imperativo: Pascal, C, Java, etc. Usaremos **Dafny**

Ejemplo: {{P}} S {{Q}}

Precondición

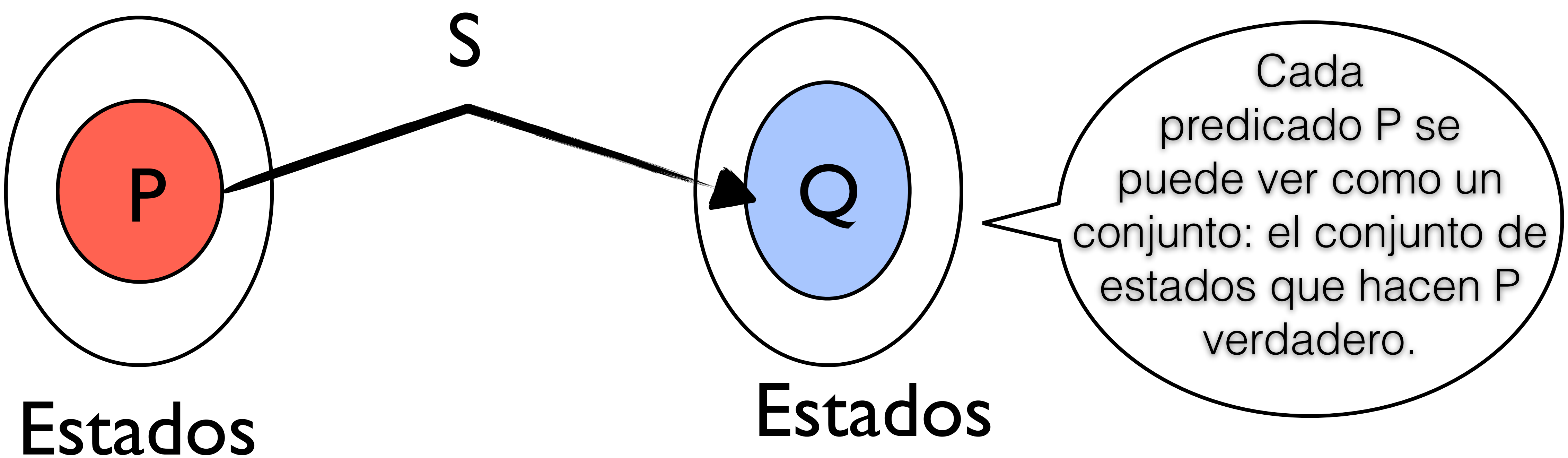
Programa

Postcondición

El programa S debe terminar satisfaciendo Q, si empieza desde estados que satisfacen P

Ternas de Hoare y Conjuntos

Podemos interpretar {{P}}S{{Q}} utilizando conjuntos:



{{P}}S{{Q}} se cumple si siempre que partimos de algún estado del conjunto P, se llega por S a un estado que pertenece a Q

Programación por Contratos

Muchas veces las especificaciones con pre y post-condiciones pueden entenderse como un contrato entre el programador y el usuario

$$\{\{P\}\}S\{\{Q\}\}$$

Es lo que el usuario debe garantizar para ejecutar el programa

Es lo que el programador garantiza si se cumple la precondición

Si la precondición no se cumple el programador “queda liberado” de cumplir la postcondición, es decir, el programa puede mostrar cualquier comportamiento

Un Ejemplo

Veamos un pequeño ejemplo:

Precondición

$\{\{x \geq 10\}\}$
```
var a := x+3;
var b := 12;
y := a+b;
```
$\{\{y \geq 25\}\}$

Postcondición

Podemos razonar paso a paso:

Este es un razonamiento para de arriba para abajo

En vez de usar estados usamos predicados para identificar conjuntos de estados.

$\{\{x \geq 10\}\}$
```
var a := x+3;
```
$\{\{x \geq 10 \wedge a = x + 3\}\}$
```
var b := 12;
```
$\{\{x \geq 10 \wedge a = x + 3 \wedge b = 12\}\}$
```
y := a+b;
```
$\{\{x \geq 10 \wedge a = x + 3 \wedge b = 12 \wedge y = a + b\}\}$

Al final tenemos este predicado

Es decir deberíamos probar:

$$(x \geq 10 \wedge a = x + 3 \wedge b = 12 \wedge y = a + b) \Rightarrow (y \geq 25)$$

Propiedades de las Ternas de Hoare

- Fortalecimiento de la precondición:

$$\{\{P\}\}S\{\{Q\}\} \wedge (P_0 \Rightarrow P) \Rightarrow \{\{P_0\}\}S\{\{Q\}\}$$

Si el programa S, cuando empieza en un estado satisfaciendo P, termina en un estado satisfaciendo Q, y Po es más fuerte que P. Entonces, si empezamos en un estado satisfaciendo Po, también garantizaremos Q

- Debilitamiento de la postcondición:

$$\{\{P\}\}\ S\ \{\{Q\}\} \wedge Q \Rightarrow Q_0 \Rightarrow \{\{P\}\}\ S\ \{\{Q_0\}\}$$

Si el programa S, cuando empieza en un estado satisfaciendo P, termina en un estado satisfaciendo Q, y Qo es más débil que Q. Entonces, si empezamos en un estado satisfaciendo P, también garantizaremos Qo.

Propiedades de las Ternas de Hoare (cont)

Conjunción de la postcondición:

$$\{\{P\}\}S\{\{Q\}\} \wedge \{\{P\}\}S\{\{Q'\}\} \equiv \{\{P\}\}S\{\{Q \wedge Q'\}\}$$

Si el programa S, cuando empieza en un estado satisfaciendo P, podemos asegurar postcondiciones Q y Q'. Entonces podemos asegurar $Q \wedge Q'$

Disyunción de la precondición:

$$\{\{P\}\}S\{\{Q\}\} \wedge \{\{P\}\}S\{\{Q'\}\} \equiv \{\{P \vee P'\}\}S\{\{Q\}\}$$

Dado un estado satisfaciendo$P$, S garantiza Q, y dado un estado satisfaciendo P', S también garantiza Q. Entonces, partiendo de un estado satisfaciendo $P \vee P'$, S también garantiza Q.

Ejemplos

Veamos los siguientes ejemplos:

$$\{\{x = 12\}\} x := x + 8 \{\{x = 20\}\}$$

Si el programa x:=x+8 comienza en un estado satisfaciendo x=12, entonces terminará en un estado satisfaciendo x=20

$$\{\{x < 18\}\} S \{\{y \geq 0\}\}$$

Cuando S empieza en un estado satisfaciendo x<18, entonces terminará en un estado satisfaciendo $y \geq 0$

Cuando una terna de Hoare se cumple (es verdadera) se dice que el programa es correcto

Hay muchas posibles implementaciones para esta especificación, por ejemplo:

$y := 5$ **ó** $y := 18 - x$ **son implementaciones correctas**

$y := x$ **ó** $y := 2 * (x + 3)$ **son implementaciones incorrectas**

Buscando la precondición

Supongamos el siguiente programa y postcondición

Que es lo mínimo que podemos pedir para que nuestro programa cumpla la postcondición

$$\{\{?\}\}$$

$$x := x + 1$$

$$\{\{x \leq y\}\}$$

Posibles candidatos son $x \leq y - 10$, $x < y$, $2 * x < y$, etc

De todas ellas la más débil (la que nos da lo mínimo que podemos pedir) es x<y

Ejemplo mucho más difícil.

$$\{\{?\}\}$$

$$x := 2 * x + y;$$

$$\{\{3 * x + 5 * y < 100\}\}$$

Qué es lo mínimo que podemos pedir para que el programa sea correcto?

Weakest Precondition (precondición más débil)

Podemos definir una función que dado un programa (texto) y una postcondición, nos devuelva la precondición más débil:

WP : Programa $\rightarrow$ Predicado $\rightarrow$ Predicado es una función que cumple:

- $\{\{WP(S)(Q)\}\}S\{\{Q\}\}$, devuelve una precondición valida.
- Para toda P tal que $\{\{P\}\}S\{\{Q\}\}$ se cumple $P \Rightarrow WP(S)(Q)$. Es decir, es la más débil.

Usaremos la notación $WP(S)(Q)$ para denotar la precondición más débil del programa S con la postcondición Q

Las ternas de Hoare y el WP

La principal virtud del WP es que nos permite demostrar si una terna de Hoare es verdadera

$$\{\{P\}\}S\{\{Q\}\} \equiv P \Rightarrow WP(S)(Q)$$

Es decir, demostrar que un programa es correcto con respecto a una pre y postcondición, es lo mismo que demostrar que la precondición implica el WP.

Probar $\{\{x < y - 10\}\}x := x + 1\{\{x < y\}\}$ es lo mismo que probar $x < y - 10 \Rightarrow WP(x := x + 1)(x < y)$

Definiendo el WP

Para cada sentencia de Dafny definiremos como calcular su WP. La más fácil es la asignación:

$$WP(x := E)(Q) = Q[x := E]$$

$Q[x := E]$ es el predicado que obtenemos cuando x lo sustituimos por E en Q

Ejemplo: $\{\{?\}\}\ x := 2 * x + y; \{\{3 * x + 5 * y < 100\}\}$

Respuesta: $6 * x + 5 * y < 100$

Asignaciones Multiples

Dafny soporta asignaciones multiples, es decir:

$$x, y := E, F$$

Los valores de x e y son actualizados al mismo tiempo

Las asignaciones múltiples nos permiten escribir código más fácilmente.

El WP es similar a caso de la asignación simple:

$$WP(x, y := E, F)(Q) = Q[x := E, y := F]$$

Utilizando el WP

Podemos usarlo para ver la corrección o bien para deducir partes del código

$$\{\{True\}\}x, y := x + 1, x + 2\{\{y = x + 1\}\}$$

Para que este código sea correcto debería pasar:

$$True \Rightarrow WP(x, y := x + 1, x + 2)(y = x + 1)$$

$\equiv$ [Def.WP]

$$True \Rightarrow x + 2 = x + 1 + 1$$

$\equiv$ [Lógica]

$$True$$

Calculando Código

Podemos utilizar el WP para encontrar partes de código.
Por ejemplo:

$$\{\{q = a * c \wedge w = c^2\}\} a, q := a + c, E \{\{q = a * c\}\}$$

Supongamos que queremos encontrar E:

$$q = a * c \wedge w = c^2 \Rightarrow WP(a, q := a + c, E)(q = a * c)$$

$\equiv$ [Def.WP]

$$q = a * c \wedge w = c^2 \Rightarrow E = (a + c) * c$$

$\equiv$ [Reemplazo]

$$q = a * c \wedge w = c^2 \Rightarrow E = a * c + c^2$$

$\equiv$ [Aritmética]

$$q = a * c \wedge w = c^2 \Rightarrow E = q + w$$

Encontramos E

Sentencias Condicionales

En Dafny la sentencia condicional (if) se escribe de la siguiente forma:

Guarda

Código que se ejecutará si la guarda es verdadera

Código que se ejecutará si la guarda es falsa

```
if B  {S1} else {S2}
```

Corrección del If:

```
{{P}}
if B{
 S1
}
else{
 S2
}
{{Q}}
```

Debemos ver:

$$1 - \{\{P \wedge B\}\} S1 \{\{Q\}\}$$

La primer rama es correcta

$$2 - \{\{P \wedge \neg B\}\} S2 \{\{Q\}\}$$

La segunda rama es correcta

El WP del If

Podemos definir el WP del If de la siguiente forma:

Sea: IF = `if B {S1} else {S2}`

Entonces:

$$WP(IF)(Q) = (B \Rightarrow WP(S1)(Q)) \wedge (\neg B \Rightarrow WP(S2)(Q))$$

B implica el WP de la primera rama

$\neg B$ implica el WP de la segunda rama

Un Ejemplo

El siguiente es un código en Dafny que calcula el máximo entre dos números.

```
{{True}}
if (x < y){
  r:=y;
}
else{
  r:=x;
}
{{r = max(x,y)}}
```

El WP de este programa es:

$$((x < y) \Rightarrow WP(r := y)(r = \max(x, y))) \wedge ((x \geq y) \Rightarrow WP(r := x)(r = \max(x, y)))$$

$\equiv$ [Def.WP]

$$((x < y) \Rightarrow y = \max(x, y)) \wedge ((x \geq y) \Rightarrow (x = \max(x, y))$$

$\equiv$ [Prop.max]

$True$

La precondición más débil es True

Por ende el programa de más arriba es correcto

Sentencias Condicionales con Multiples Guardas

Cuando tenemos muchas opciones, podemos escribir expresiones con multiples guardas:

```
if{
 case B1 => S1
 case B2 => S2
  . . .
 case Bn => Sn
}
```

n ramas

Cada rama se compone de:

```
B => S
```

Guarda

Código asociado

Si hay más de una rama verdadera se elige una no-deterministicamente

Ejemplo:

El if tiene que ser exhaustivo

```
if{
  case x <= z && y <= z => r := z;
  case x <= y && z <= y => r := y;
  case y <= x && z <= x => r := x;
}
```

Máximo entre tres números

Corrección de Sentencias Condicionales

Es similar a los ifs, pero se tiene que verificar que las guardas son exhaustivas:

```
{{P}}
if{
  case B1 => S1;
  case B2 => S2;
  . . .
  case Bn => Sn;
}
{{Q}}
```

tenemos que probar:

- $B1 \vee \ldots \vee Bn$
- $Bi \Rightarrow WP(Si)(Q)$

Para todo i

El WP es similar al If común, teniendo en cuenta los casos.

Los Ciclos

En Dafny tenemos varias alternativas para iteraciones, uno de los más útiles es el *while:*

```
while B{
    S
}
```

Mientras B es verdadero, se ejecuta S

Al igual que con el if tenemos una versión con muchas guardas:

```
while{
   case B1 => S1
   case B2 => S2
   . . .
}
```

Cuando todas las guardas son falsas, se sale del while. En otro caso, se elige algunas de las guardas verdaderas y se ejecuta

Invariantes

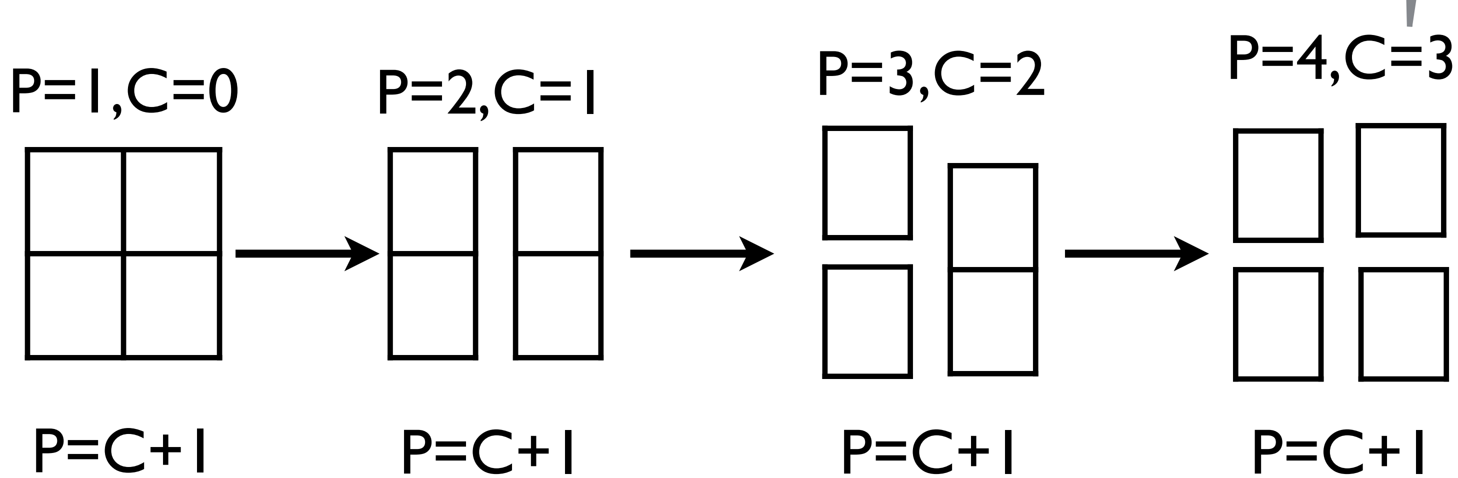
Para demostrar la corrección de un while necesitamos la noción de invariante:

Ejemplo de la barra de chocolate:
- Un pedazo de chocolate
- N bloques
- Cuantos cortes necesitamos?

Un invariante nos permite razonar sobre una iteración

$$
\begin{aligned}
& P = N \\
\equiv\ & [\text{Invariante}] \\
& C + 1 = N \\
\equiv\ & [\text{Aritmética}] \\
& C = N - 1
\end{aligned}
$$

Necesitamos N-1 cortes



Corrección de un Ciclo

Para demostrar la corrección de un ciclo:

```
{{P}}
S0
while B{
   S
}
{{Q}}
```

Precondición

Inicialización

Guarda

Código

Postcondición

Esto es **corrección parcial** no decimos nada de la terminación

Tenemos que encontrar un predicado I tal que:

- $\{\{P\}\}S0\{\{I\}\}$ — La inicialización hace verdadero el invariante
- $\neg B \wedge I \Rightarrow Q$ — Al finalizar el ciclo el invariante nos permite asegurar la postcondición
- $\{\{B \wedge I\}\}S\{\{I\}\}$ — El invariante se mantiene durante el ciclo

Un Ejemplo

Veamos un ejemplo:

$\{\{n > 0 \wedge m > 0\}\}$

```
var i,r := 0,1;
while i<m{
    i := i+1;
    r := r*n;
}
```

$\{\{r = n^m\}\}$

Calcula n^m

Para buscar un invariante podemos ver que relación hay entre las variables:

	i	r
no entra al ciclo	0	1
primera it.	1	n
segunda it.	2	n*n
tercera it.	3	n*n*n

Un buen candidato es $r = n^i$, también se cumple $i \leq m$, toda información útil la podemos agregar en el invariante

Corrección

Para verificar la corrección del ejemplo anterior debemos comprobar:

$$1.\{\{n > 0 \wedge m > 0\}\} i, r := 0,1 \{\{r = n^i \wedge i \leq m\}\}$$

$$2. r = m^i \wedge i \leq m \wedge i \geq m \Rightarrow r = n^m$$

$$2.\{\{r = n^i \wedge i \leq m \wedge i < m\}\} r, i := r * n, i + 1 \{\{r = n^i \wedge i \leq m\}\}$$

Si comprobamos todas estas condiciones (usando WP) nuestro programa es **parcialmente correcto**.

Corrección Total

Para demostrar la corrección total (es decir, que el programa termina) debemos encontrar un **variante** (o cota, o función de decremento).

Un variante es una expresión entera V tal que:

- $V = 0 \wedge I \Rightarrow \neg B$

Cuando V llega a 0 se sale del ciclo

- $\{\{I \wedge B \wedge V = A\}\}S\{\{V < A\}\}$

Cada iteración decrementa V

Ejemplo

Veamos el ejemplo de las potencias:

$\{\{n > 0 \wedge m > 0\}\}$

```
var i,r := 0,1;
while i<m{
    i := i+1;
    r := r*n;
}
```

$\{\{r = n^m\}\}$

En cada iteración i se acerca a m

Cuando i = m se sale del ciclo, y el invariante nos da la poscondición

Es decir, un buen candidato para el variante es: $m - i$

Encontrando Invariantes

Los invariantes son, en general, versiones más débiles de la postcondición.