Programación Imperativa

Programación Avanzada-UNRC
Pablo Castro

Estados y Predicados

- La programación imperativa se basa en la noción de estado y variables.
- Cada programa imperativo utiliza un conjunto de variables: x, y, z, ...
- Un estado en la ejecución de un programa imperativo es una asignación de valores a sus variables.
- Las instrucciones nos permiten cambiar de estados.

```
program example(output);
var i : integer;
var j : integer;
....
```

Variables: i,j

Posible Estado:
{(i,2),(j,3)}

Especificando Programas Imperativos

- A Diferencia de funcional la ejecución de un programa imperativo produce cambios de estado.
- En este caso una especificación indica el estado en que esperamos que termine un programa (**postcondición**), dada cierta condición inicial (**precondición**).

Escribimos una especificación de la siguiente forma:

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

Precondición

Programa

Postcondición

Si ejecutamos S comenzando en un estado que satisface P, se termina en un estado que satisface Q

Pre y Postcondiciones

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

Ejemplo:

$$\{True\}S\{x=y\}$$

El programa S debe terminar satisfaciendo x=y, para cualquier valor de x e y

Estados y Predicados

Dado un estado s, diremos que se cumple:

$$s \models P$$

Cuando los valores asignados por s a las variables hacen P verdadero.

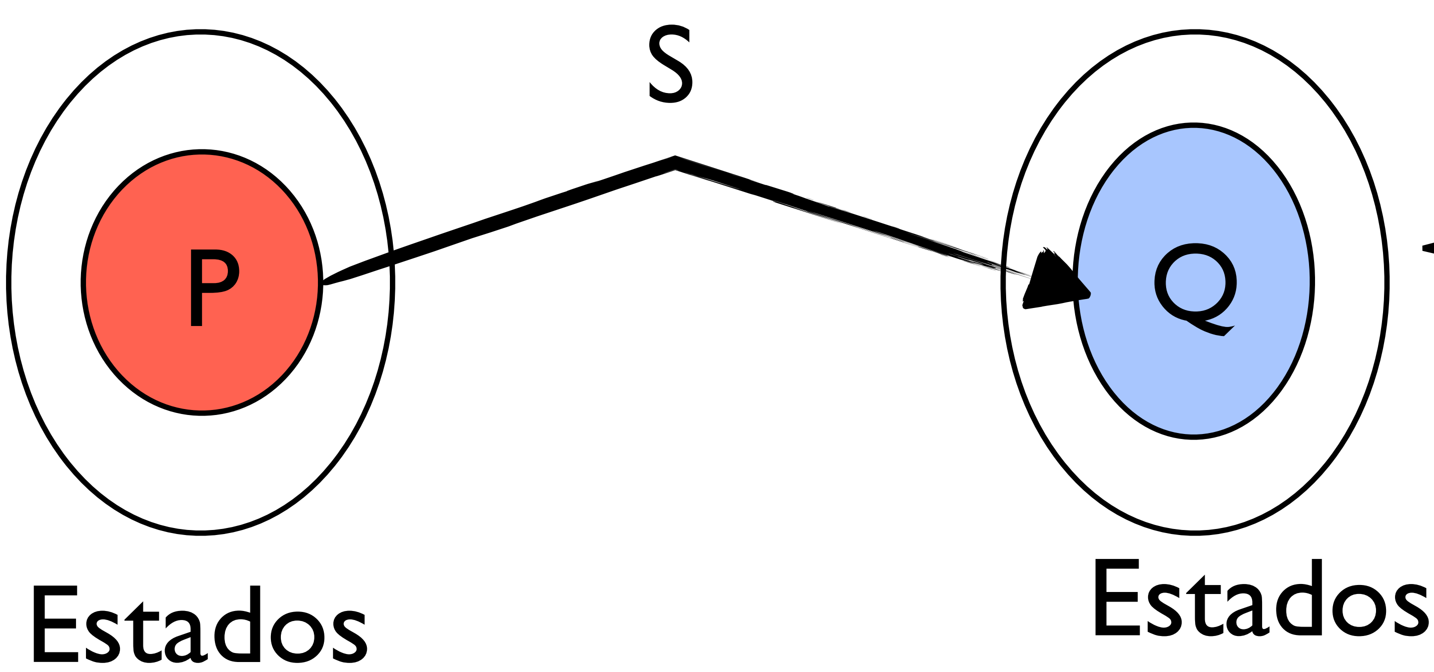
$$\{(x,5),(y,10)))\} \models y = x + 5$$

Estado

Predicado

Ternas de Hoare y Conjuntos

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



Cada predicado P
se puede ver como un
conjunto: el conjunto de
estados que hacen P
verdadero.

{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

Predicados Universalmente Validos

Diremos que:

$$[P]$$

P es un predicado

Cuando:

$$\langle \forall s : s \text{ es estado} : s \models P \rangle$$

Todo estado satisface P

Por ejemplo:

$$[0 = 1 \Rightarrow x = y]$$

Es verdadero independientemente de x e y

Ternas de Hoare

Una terna de Hoare es:

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

Estas ternas cumplen ciertas propiedades lógicas:

- Exclusión de milagros: $\{P\}S\{False\} \equiv [P \equiv False]$

No hay ningún estado del cual podamos empezar, y terminar satisfaciendo False

Propiedades de la 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

Ejemplo:

$$\{x = 2 \vee x = 5\}S\{Q\} \Rightarrow \{x = 2\}S\{Q\}$$

- 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.

Ejemplo:

$$\{P\}S\{x = 2\} \Rightarrow \{P\}S\{x = 2 \vee x = 5\}$$

Propiedades de la ternas de Hoare (cont.)

- Conjunción de 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^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 PvP', S también garantiza Q.

El Transformador de Predicados *wp*

Para cada comando S, podemos definir una función

$$wp.S : Predicados \rightarrow Predicados$$

Dado un predicado Q, wp.S devuelve el predicado (precondición) más débil que cumple:

{WP. S. Q} S {Q}

Es decir, wp.S.Q cumple:

$$1. \{wp.S.Q\}S\{Q\}$$

$$2. \langle \forall P' :: \{P'\}S\{Q\} \Rightarrow (P' \Rightarrow wp.S.Q) \rangle$$

El Transformador de predicados *wp*(cont)

La función wp nos permite probar {P}S{Q}:

$$\{P\}S\{Q\} \equiv [P \Rightarrow wp.S.Q]$$

Usando esta propiedad podemos decir que un programa {P}S{Q} es correcto cuando se cumple: *[P => wp.S.Q]*

Podemos escribir las reglas anteriores de la siguiente forma:

$$1.[wp.S.False \equiv False]$$

$$2.[wp.S.Q \wedge wp.S.R \equiv wp.S.(Q \wedge R)]$$

$$3.[wp.S.Q \vee wp.S.R \Rightarrow wp.S.(Q \vee R)]$$

Un pequeño lenguaje Imperativo

Tendremos 3 clases de variables:

- *Constantes*: sirven para hablar de los valores de las variables.
- *Variables* de programación.
- *Variables* cuantificadas.

```
begin
    cons X,Y: Int;
    var x,y:Int;
```

$\{X > 0 \wedge Y > 0 \wedge x = X \wedge y = Y\}$

```
    S
```

$\{x = mcd.X.Y\}$

```
end
```

La sentencia *skip*

La sentencia skip nos devuelve el mismo estado:

- $wp.skip.Q \equiv Q$
- $\{P\}skip\{Q\} \equiv [P \Rightarrow Q]$

El programa:

$$\begin{array}{l} begin \\ \{x \geq 1\} \\ skip \\ \{x \geq 0\} \\ end \end{array}$$

es correcto ya que:

$$[x \geq 1 \Rightarrow x \geq 0]$$

El skip tiene la misma utilidad que el 0 en aritmética!

La Asignación

La asignación es una de las sentencias más importantes:

$$x := E$$

Variable de Programación

Expresión del mismo tipo que x

El efecto de un asignación es cambiar el valor de una variable:

$$\{\ldots, x \mapsto 5, y \mapsto 1, \ldots\} \xRightarrow{x := y + 2} \{\ldots, x \mapsto 3, y \mapsto 1, \ldots\}$$

La Asignación (cont.)

Lo mínimo que requerimos para que Q sea verdadero después de S es $Q[x := E]$.

Sustitución

$$[wp.(x := E).Q \equiv Q(x := E)]$$

Es decir, para que $\{P\}S\{Q\}$ sea verdadero, P debe ser más fuerte que $Q[x := E]$:

$$\{P\}x := E\{Q\} \equiv [P \Rightarrow Q[x := E]]$$

También utilizaremos la asignación de varias variables:

$$x, y, z := x + 1, y * 2, z + 10$$

Ejemplo

Supongamos que queremos demostrar:

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

Veamos:

$[true \Rightarrow (y = x + 1)[x, y := x + 1, x + 2]]$

$\equiv$ [Lógica]

$[(y = x + 1)[x, y := x + 1, x + 2]]$

$\equiv$ [Sustitución]

$x + 2 = x + 1 + 1$

$\equiv$ [Arit.]

$true$

La composición

La composición (o secuencia) nos permite escribir secuencias de acciones:

$$S; S'$$

Primero se ejecuta S y luego se ejecuta S'

Para probar una composición, tenemos que encontrar un predicado intermedio:

$$\{P\}S; S'\{Q\} \equiv \exists R : \{P\}S\{R\} \wedge \{R\}S'\{Q\}$$

Para calcular el wp de $S; S'$, primero calculamos el wp de S' y luego la precondición de S

$$wp.(S; S').Q \equiv wp.S.(wp.S'.Q)$$

Ejemplo:

Probemos: $\{x > y\}x := x + 1; y := y + 1\{x > y\}$

$x > y \Rightarrow wp.(x := x+1; y := y+1).(x > y)$

$\equiv$ [def. wp]

$x > y \Rightarrow wp.x := x+1.(wp.y := y+1.(x > y))$

$\equiv$ [def. wp]

$x > y \Rightarrow wp.x := x + 1.(x > y + 1)$

$\equiv$ [Aritmética]

$x > y \Rightarrow (x + 1 > y + 1)$

También podemos encontrar un predicado intermedio para demostrar la corrección:

$$\{x > y\}x := x + 1; \{x > y + 1\}x := y + 1\{x > y\}$$

La Sentencia "If"

Una de las sentencias importantes es la alternativa, nos permite definir alternativas de ejecución:

$$
\begin{array}{l}
if\ B_0 \rightarrow S_0 \\
\Box\ B_1 \rightarrow S_1 \\
\quad \vdots \\
\Box\ B_n \rightarrow S_n \\
fi
\end{array}
$$

De todos los comandos S_i cuya guarda B_i es verdadera, se elige uno de forma no-determinista y se lo ejecuta

Comandos

Guardas

Este programa devuelve:
$x = cara$
ó
$x = seca$

$$
\begin{array}{l}
\{True\} \\
if\ True \rightarrow x := cara \\
\Box\ True \rightarrow x := seca \\
\{x = cara \vee x = seca\}
\end{array}
$$

La Sentencia "If" (cont)

Para probar la corrección:

- Se debe probar que alguna guarda es verdadera.
- Se debe probar que todas las alternativas son correctas.

$$
\begin{array}{l}
\{P\} \\
if\ B_0 \to S_0 \\
\ \square\ B_1 \to S_1 \\
\quad \vdots \\
\ \square\ B_n \to S_n \\
\{Q\}
\end{array}
\equiv
\begin{array}{l}
[P \Rightarrow B_0 \vee \cdots \vee B_n \\
\wedge \\
\{P \wedge B_0\} S_0 \{Q\} \\
\wedge \\
\vdots \\
\wedge \\
\{P \wedge B_n\} S_n \{Q\}]
\end{array}
$$

Se ejecuta al menos una guarda

Cada alternativa es correcta

En cuanto al wp:

$$wp.if.Q \equiv [(B_0 \vee B_1 \vee \ldots \vee B_n) \wedge (B_0 \Rightarrow wp.S_0.Q) \wedge \ldots \wedge (B_n \Rightarrow wp.S_n.Q)]$$

Probando la corrección del If

Para demostrar un if, tenemos:

$$\text{I. } [P \Rightarrow B_0 \vee \cdots \vee B_n]$$

$$\text{II. } [P \wedge B_i \Rightarrow wp.S_i.Q]$$

Para toda rama del IF

Por ejemplo:

$[x = X \wedge y = Y \Rightarrow x < y \vee x \geq y]$

$\{x = X \wedge y = Y\}$

$if\ x < y \ \rightarrow x := y$

$[x < y \wedge (x = X \wedge y = Y) \Rightarrow wp.(x := y).((x = X \vee x = Y) \wedge x \geq X \wedge x \geq Y$

$\Box\ x \geq y \rightarrow skip$

$[x \geq y \wedge (x = X \wedge y = Y) \Rightarrow wp.(skip).((x = X \vee x = Y) \wedge x \geq X \wedge x \geq Y)]$

$\{(x = X \vee x = Y) \wedge x \geq X \wedge x \geq Y\}$

Derivando Programas

Consideremos el siguiente programa:

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

Encontremos un valor para E:

$$[q = a*c \wedge w = c^2 \Rightarrow wp.(a, q := a+c, E).(q = a*c)]$$

$$\equiv [\text{Def.wp}]$$

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

$$\equiv [\text{Aritmética}]$$

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

$$\equiv [Leibniz]$$

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

$$\Leftarrow [\text{Lógica}]$$

$$E = q + w$$

Encontramos E!

Calculando If's

Si tenemos una especificación: $\{P\}S\{Q\}$ podemos desarrollar un *If* cuando:

$$1. P \Rightarrow B_0 \vee \cdots \vee B_n, \text{ para algunos predicados } B_i$$

$$2. \{P \wedge B_i\}S_i\{Q\}, \text{ para cada } i$$

En este caso obtendremos el programa:

$$\begin{array}{l} if B_0 \rightarrow S_0 \\ \vdots \\ \Box B_n \rightarrow S_n \\ fi \end{array}$$

Equivalencia de Programas

Dos programas S y T se dicen equivalentes ($S = T$) cuando:

$$\langle \forall Q :: wp.S.Q = wp.T.Q \rangle$$

Es decir, cuando sus precondiciones más débiles son siempre las mismas. Ejemplos:

$$x := x = skip$$

$$skip; S = S; skip$$

Ejercicio: Demostrar estas igualdades

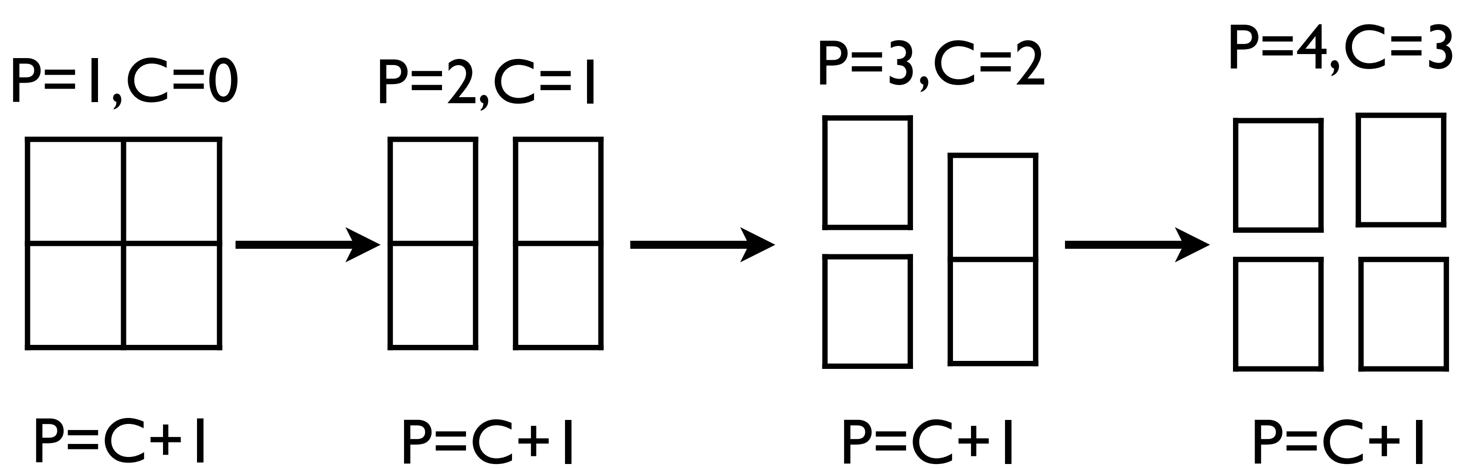
Invariantes

Los invariantes nos permiten razonar sobre ciclos:

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

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

Necesitamos N-1 cortes



Repetición

La repetición nos permite ejecutar de forma repetida una acción

$$
\begin{array}{l}
\{P\} \\
do\ B_0 \rightarrow S_0 \\
\Box\ B_1 \rightarrow S_1 \\
\quad\quad \vdots \\
\Box\ B_n \rightarrow S_n \\
od \\
\{Q\}
\end{array}
$$

Guardas

Comandos

Un do elige en cada paso una guarda verdadera (de una forma no-determinista), y ejecuta el comando correspondiente. Procede así, hasta que todas las guardas son falsas.

Para demostrar la corrección debemos encontrar I tal que:

$$
\begin{array}{l}
I \wedge \neg B_0 \wedge \cdots \wedge \neg Bn \Rightarrow Q \\
\wedge\{I \wedge B_0\}S_0\{I\} \\
\vdots \\
\wedge\{I \wedge B_n\}S_0\{I\} \\
\text{y el ciclo termina}
\end{array}
$$

Cuando termina el ciclo vale la postcondición

I se llama Invariante

Un Ejemplo

Analicemos el siguiente programa:

Inicialización

$$\{n \geq 0 \wedge m \geq 0\}$$
$$r, i := 1, 0;$$
$$do\ i < m \rightarrow r, i := r * n, i + 1;\ od$$
$$\{r = n^m\}$$

Postcondición: El programa calcula en r el valor de n elevado a la m

En cada paso del ciclo tenemos calculado:

Invariante

$$r = n^i$$

y

$$i \leq m$$

Este predicado es un buen candidato para invariante. Cuando el ciclo termina tenemos i=m, lo cual hace verdadera la postcondición

Terminación

La variable i en cada paso del do se acerca más al m.

$$\{P \wedge m - i = A\} r, i := r * n, i + 1 \{P \wedge m - i < A\}$$

Esto nos garantiza que el ciclo termina, el valor m-i es siempre más chico, pero tiene que parar cuando m-i=0

Además tenemos:

$$i < m \Rightarrow 0 < m - i$$

Estas expresiones son llamadas *variantes*, deben cumplir:

$$P \wedge B_i \Rightarrow v > 0$$

Durante el ciclo el valor es positivo

$$\{P \wedge B_i \wedge v = A\} S_i \{v < A\}$$

En cada paso el valor del variante se decrementa

Teorema del *do*

Para demostrar la corrección de un *do* debemos demostrar los siguientes puntos:

- Inicialización: $\{P\}S\{I\}$
  La inicialización hace verdad el invariante
- Postcondición: $I \wedge \neg B_0 \wedge \cdots \wedge \neg B_n \Rightarrow Q$
  Cuando termina el ciclo vale la postcondición
- :Invariante: $\{B_i \wedge I\}S_i\{I\}$
  "I" es un invariante
- Variante (a): $I \wedge B_i \Rightarrow v \geq 0$
  Adentro del *do* "v" es siempre positivo
- Invariante (a): $\{I \wedge B_i \wedge v = A\}S_i\{v < A\}$
  Cada paso del *do* decrementa "v"

Variantes

Es importante notar que la formula:

$$I \wedge B_i \Rightarrow v > 0$$

Se puede escribir de varias formas equivalentes:

$$I \wedge B_i \Rightarrow v \geq 0$$

$$I \wedge v < 0 \Rightarrow \neg B_i$$

Ejercicio: Demostrar la equivalencia de estas formulas!

Ejemplo

Para el caso anterior tenemos que demostrar:

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

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

$$3.\{r = n^i \wedge i \leq m \wedge i < m\} r, i := r * n, i + 1; \{r = n^i \wedge i \leq m\}$$

$$4. r = n^i \wedge i \leq m \wedge i < m \Rightarrow m - i > 0$$

$$5.\{r = n^i \wedge i \leq m \wedge m - i = A\} r, i := r * n, i + 1; \{m - i < A\}$$

Y el programa el correcto!

Otro Ejemplo

Suma de un arreglo:

$$con\ N : Int; a : Array[0, N)\ of\ Int;$$
$$var\ x, n : Int;$$
$$x, n := 0, 0;$$
$$\{P : N \geq 0\}$$
$$do\ n < N \rightarrow x, n := x + a.n, n + 1;\ od$$
$$\{Q : x = \langle \sum i : 0 \leq i < N : a.i \rangle\}$$

En cada paso del ciclo tenemos en x calculado la suma a[0..n]

Podemos poner como invariante:

$$\{I : (x = \langle \sum i : 0 \leq i < n : a.n \rangle) \wedge n \leq N\}$$