Especificación formal de problemas

Yolanda Ortega Mallén
(basado en documento de I. Pita)

Dpto. de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

Bibliografía

- N. Martí Oliet, C. Segura Díaz y J. A. Verdejo López.
  Algoritmos correctos y eficientes: diseño razonado ilustrado con ejercicios.
  Garceta Grupo Editorial, 2012. Capítulo 1.

Sintaxis de la lógica de primer orden:

Un predicado es una expresión e de tipo *bool*.

Si P y Q son predicados, también lo son:

$\neg P$ no P.

$P \wedge Q$ P y Q.

$P \vee Q$ P o Q.

$P \rightarrow Q$ P entonces Q.

$P \leftrightarrow Q$ P si y solo si Q.

Cuantificación universal: $(\forall w : Q(w) : P(w))$ para todo valor perteneciente al rango Q se cumple P.

Cuantificación existencial: $(\exists w : Q(w) : P(w))$ existe un valor perteneciente al rango Q para el que se cumple P.

Ejemplos de predicados

- $n \geq 0$
- $x > 0 \land x \neq y$
- $0 \leq i < n$ $\{$abreviatura de $0 \leq i \land i < n\}$
- $\forall w : 0 \leq w < n : a[w] \geq 0$
- $\exists w : 0 \leq w < n : a[w] = x$
- $\forall w : 0 \leq w < n : impar(w) \rightarrow (\exists u : u \geq 0 : a[w] = 2^u)$
  donde $impar(x)$ es un predicado que dice si su argumento x es impar o no.

Variables ligadas Están dentro del ámbito de un cuantificador.

Variables libres No se encuentran afectadas por ningún cuantificador. Describen variables del programa.

- Cuantificadores anidados: variables ligadas al cuantificador más interno.
- Una variable ligada se puede renombrar sin cambiar el significado del predicado.

$$(\exists w : 0 \leq w < n : a[w] = x) \equiv (\exists v : 0 \leq v < n : a[v] = x)$$

Utilizar identificadores diferentes en las variables ligadas para evitar errores. Utilizar identificadores diferentes para las variables libres de los usados en las variables ligadas.

$\sum w : Q(w) : e(w)$	suma de las $e(w)$ tales que $Q(w)$.
$\prod w : Q(w) : e(w)$	producto de las $e(w)$ tales que $Q(w)$.
$\max w : Q(w) : e(w)$	máximo de las $e(w)$ tales que $Q(w)$.
$\min w : Q(w) : e(w)$	mínimo de las $e(w)$ tales que $Q(w)$.
$\# \, w : Q(w) : P(w)$	número de valores que verifican $P(w)$ de entre los que cumplen $Q(w)$.

donde Q y P son predicados y e es una expresión entera.

Utilizaremos predicados para definir conjuntos de estados.

Estado

Asociación de las variables del algoritmo con valores compatibles con su tipo.

- Un estado σ satisface un predicado P si al sustituir en P las variables libres por los valores que esas variables tienen en σ el predicado evalúa a cierto.
- Identificaremos un predicado con el conjunto de estados que lo satisfacen.
- P es un predicado más fuerte que Q: $P \Rightarrow Q$
  si el conjunto de estados que satisfacen P es un subconjunto del de estados que satisfacen Q: $P \subseteq Q$.
  P es un predicado más débil que Q: $P \Leftarrow Q$
  si el conjunto de estados que satisfacen P es un superconjunto del de estados que satisfacen Q: $P \supseteq Q$.
- P y Q son equivalentes: $P \equiv Q$,
  cuando $P \Rightarrow Q$ y $P \Leftarrow Q$.
  Dos predicados equivalentes definen el mismo conjunto de estados.

- El predicado `cierto` indica que las variables pueden tomar cualquier valor; define el conjunto de todos los estados posibles.
- El predicado `falso` no se cumple para ningún valor de las variables; define el conjunto vacío.
  No tiene sentido su uso en la precondición;
  en la postcondición indica que el programa no termina.
- $\forall w : Q(w) : P(w)$ se corresponde con $P(w_1) \wedge P(w_2) \wedge \ldots$,
  donde $w_1, w_2, \ldots$ son todos los valores de w que hacen cierto $Q(w)$.
  Si este conjunto es vacío, entonces $\forall w : Q(w) : P(w) \equiv \texttt{cierto}$.
- $\exists w : Q(w) : P(w)$ se corresponde con $P(w_1) \vee P(w_2) \vee \ldots$,
  donde $w_1, w_2, \ldots$ son todos los valores de w que hacen cierto $Q(w)$.
  Si este conjunto es vacío, entonces $\exists w : Q(w) : P(w) \equiv \texttt{falso}$.
- Por definición, cuando el rango $Q(w)$ al que se extiende la variable cuantificada es *vacío*:

$$
\begin{array}{ll}
(\sum w : \texttt{falso} : e(w)) = 0 & \\
(\prod w : \texttt{falso} : e(w)) = 1 & \\
(\text{máx}\, w : \texttt{falso} : e(w)) & \text{indefinido} \\
(\text{mín}\, w : \texttt{falso} : e(w)) & \text{indefinido} \\
(\#\, w : \texttt{falso} : P(w)) = 0 &
\end{array}
$$

- Identificamos un algoritmo con una función o un procedimiento:

  fun `nombre`$(p_1 : \textit{tipo}_1, \ldots, p_n : \textit{tipo}_n)$ **dev** $r : \textit{tipo}$
  proc `nombre`$(\textit{cualif}\ p_1 : \textit{tipo}_1, \ldots, \textit{cualif}\ p_n : \textit{tipo}_n)$

- Los parámetros pueden ser de:

entrada	Su valor inicial es relevante para el algoritmo y éste no debe modificarlo: *cualif* vacío.
salida	Su valor inicial es irrelevante para el algoritmo y éste debe almacenar algún valor en él: *cualif* **S**.
entrada/salida	Su valor inicial es relevante para el algoritmo y además este puede modificarlo: *cualif* **E/S**.

Los parámetros de una cabecera **fun** se entienden siempre de entrada.

Devolver un número primo mayor o igual que un cierto valor

$\{\ n > 1\ \}$
fun `unPrimo`$(n : ent)$ **dev** $p : ent$
$\{\ p \geq n \land (\forall w : 1 < w < p : p \mod w \neq 0)\ \}$

- ¿Sería correcto devolver $p = 2$?
- La postcondición no determina un único p.

Devolver el menor número primo mayor o igual que un cierto valor

$\{\ n > 1\ \}$
fun `menorPrimo`$(n : ent)$ **dev** $p : ent$
$\{\ p \geq n \land primo(p) \land (\forall w : w \geq n \land primo(w) : p \leq w)\ \}$

donde $primo(x) \equiv (\forall w : 1 < w < x : x \mod w \neq 0)$.

- El predicado auxiliar $primo(x)$ hace la especificación más modular y legible.
- $primo(x)$ es una propiedad y **no** sugiere que la comprobación haya de hacerse dividiendo por todos los números menores que x.
- La postcondición puede ser más concisa:
  $p = (\min w : w \geq n \land primo(w) : w)$.

Ejemplo de especificación

Positivizar un vector. Primer intento

$\{ N \geq 0 \}$
proc `positivizar`(**E/S** $v[0..N)$ **de** *ent*)
$\{ \forall i : 0 \leq i < N : v[i] < 0 \rightarrow v[i] = 0 \}$

¡Hemos conseguido una postcondición equivalente a `falso`!

v en la postcondición se refiere a su valor después de ejecutarlo.

Positivizar un vector. Segundo intento

$\{ N \geq 0 \land v = V \}$
proc `positivizar`(**E/S** $v[0..N)$ **de** *ent*)
$\{ \forall i : 0 \leq i < N : V[i] < 0 \rightarrow v[i] = 0 \}$

La condición $v = V$ (en la precondición) sirve para dar un nombre al valor del vector v antes de ejecutar el procedimiento.

¡El implementador malévolo podría modificar también el resto de valores!

Positivizar un vector. Tercer intento

$\{ N \geq 0 \land v = V \}$
proc `positivizar`(**E/S** $v[0..N)$ **de** *ent*)
$\{ \forall i : 0 \leq i < N : (V[i] < 0 \rightarrow v[i] = 0) \land (V[i] \geq 0 \rightarrow v[i] = V[i]) \}$