CRA. Tema 5: Semántica y verificación de programas (primera parte)

José Enrique Morais San Miguel

Universidad de Alcalá

C. A. R. Hoare: “Computer programming is an **exact science** in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely **deductive reasoning**”.

¿Qué es la verificación formal?

Wikipedia: Formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

¿Por qué la verificación formal?

- Escribir código correcto no es fácil.
- Hacer tests es fácil.
- Difícil, haciendo tests, saber si existen comportamientos anómalos.
- Los tests solamente se pueden llevar a cabo sobre un número finito de casos.

Un sistema de verificación formal: Hoare Logic

- Se aplica a lenguajes imperativos
- Primer sistema propuesto
- Ejemplifica los principios generales de cualquier sistema
- Palabra clave: invariante

Un sistema de verificación formal: Hoare Logic

Principio general: Un programa no es muy diferente de una fórmula lógica. Es una secuencia de símbolos que se construye en base a reglas sintácticas formales y cuyo significado se comprende mediante una interpretación adecuada de los elementos del lenguaje. En programación, los símbolos se llaman expresiones o comandos y la correspondiente interpretación es una ejecución en una máquina, más que una evaluación de la tabla de verdad. Normalmente, la sintaxis de los lenguajes de programación se especifa mediante el uso de sistemas formales, pero la semántica se suele especificar de manera informal. Ahora bien. Si la semántica se define de manera informal, no hay manera de determinar la validez o corrección de un programa y nos vemos reducidos a testar y depurar, lo que es poco fiable.

Un sistema de verificación formal: Hoare Logic

Hay una diferencia esencial entre los programas y el cálculo de predicados. El cálculo de predicados se centra en interpretaciones arbitrarias y, en consecuencia, en las fórmulas válidas. La mayoría de programas, sin embargo, tratan con dominios numéricos u otros dominios predefinidos (cadenas, listas, árboles,...). Para razonar sobre los programas, debemos tomar los teoremas en esos dominios como axiomas. Así, la completitud de sistema axiomático para la verificación de programas depende de la teoría del dominio.

Definición: *Si un programa usa n variables* $(\mathtt{x}_1, \ldots, \mathtt{x}_n)$*, un estado s consiste en una n−tupla de de valores* $(x_1, \ldots, x_n)$*, donde* x_i *es el valor de la variable* $\mathtt{x}_i$.

Como queremos razonar dentro del marco del cálculo de predicados, debemos sustituir los conjuntos de estados por predicados.

Definición: *Sea U el conjunto de todas las n−tuplas en un cierto dominio y sea* $U' \subseteq U$*. El* **predicado característico** *de* U'*, denotado por* $P_{U'}(x_1, \ldots, x_n)$*, se define mediante:*

$$U' = \{(x_1, \ldots, x_n) \in U : P_{U'}(x_1, \ldots, x_n)\}.$$

Ejemplo: *Sea S la expresión* `x := 2*y+1`. *Esta expresión transforma estados en* $\{(x,y) \mid true\}$ *en estados en* $\{(x,y) \mid x = 2y+1\}$. *Supongamos que el conjunto de estados iniciales es* $\{(x,y) \mid y \leq 3\}$. *Tras ejecutar S, el estado final está en* $\{(x,y) \mid (x \leq 7) \wedge (y \leq 3)\}$. *En este sentido, se dice que S transforma* $y \leq 3$ *en* $(x \leq 7) \wedge (y \leq 3)$.

Toda expresión, grupo de expresiones o un programa completo puede verse como una transformador de predicados.

Un **aserto** o **terna de Hoare** es una terna $\{p\}\ S\ \{q\}$, donde S es un programa, y p y q son fórmulas del cálculo de predicados llamadas **precondición** y **postcondición**, respectivamente.

Se dice que S es **parcialmente correcto** con respecto a p y q si, y sólo si, siempre que S empiece en un estado que satisfaga p y la computación termine, ésta termina en un estado que satisface q.

Ejemplo: el aserto $\{y \leq 3\}$ `x:= 2*y+1` $\{(x \leq 7) \wedge (y \leq 3)\}$ es verdadero.

Precondiciones más débiles

La formalización de la semántica de un lenguaje en términos de predicados se da estableciendo, para cada expresión y postcondición, la mínima precondición que garantiza la verdad de la postcondición bajo la hipótesis de la finalización de la computación de la expresión. En este caso, mínima quiere decir que es aquella precondición que es verificada por mayor número de estados. Por ejemplo, $p \equiv (y \leq 3)$ no es, desde luego, la única precondición para la que $\vDash p$ `x:= 2*y+1` $\{(x \leq 7) \wedge (y \leq 3)\}$. Por ejemplo, podríamos tomar $p \equiv (y = 0)$. No es difícil ver que la precondición $(y \leq 3)$ es la que nos da el mayor número de estados iniciales posibles para los que, acabada la ejecución de `x:= 2*y+1` se obtiene un estado que satisface $\{(x \leq 7) \wedge (y \leq 3)\}$.

Definición Una fórmula A se dice más débil que B si $B \rightarrow A$. Dado un conjunto de fórmulas $\{A_1, A_2, \ldots\}$, A_i es la fórmula más débil del conjunto si $A_j \rightarrow A_i$ para todo j.

Precondiciones más débiles

Observación: Si $p_1 \longrightarrow p$ y $\{p\}\ S\ \{q\}$ es verdadero, también lo es $\{p_1\}\ S\ \{q\}$. Igualmente si $q \longrightarrow q_1$ y $\{p\}\ S\ \{q\}$ es verdadero, también lo es $\{p\}\ S\ \{q_1\}$.

Así pues, si un aserto es verdadero, fortalecer la precondición o debilitar la postcondición hacen que el aserto siga siendo verdadero.

Definición: *Para un programa S y una fórmula q se define la* **precondición más débil** *para S y q, y se denota por* $wp(S, q)$*, a la fórmula más débil p de entre las que verifican que el aserto* $\models \{p\}\ S\ \{q\}$ *es verdadero*.

Precondiciones más débiles

La precondición más débil depende tanto del programa *S* como de la postcondicióon q. Así pues, *wp* transforma predicados ya que, dado un fragmento de un programa, define una transformación de la postcondición en un predicado (la precondición más débil). Por lo tanto, en vez de formalizar un programa como una transformación de estados, lo haremos como una transformación de predicados: del predicado postcondición al predicado precondición. Esta formulación permite comenzar con la especificación del resultado del programa completo y trabajar hacia atrás para diseñar o verificar un programa que consiga el resultado fijado.

Precondiciones más débiles

$wp(\mathtt{x} := \mathtt{t}, p(x)) = p(x)\{x \leftarrow t\}$

A primera vista, la semántica de la asignación parece extraña, pero debe entenderse en términos de transformación de predicados. Si la sustitución t por x hace $p(x)$ verdadera ahora, sigue adelante y realiza la asignación, tras lo que $p(x)$ será verdadero y x tendrá valor t.

Ejemplo: $wp(\mathtt{y} := \mathtt{y} - 1, y \geq 0) = (y - 1 \geq 0) \equiv (y \geq 1)$

Precondiciones más débiles

$wp(S_1; S_2, q) = wp(S_1, wp(S_2, q))$

La precondición $wp(S_2, q)$ caracteriza el conjunto de estados tales que una ejecución de S_2 lleva a un estado en el que q es verdadero. Si la ejecución de S_1 lleva a uno de esos estados, entonces $S_1; S_2$ lleva a un estado cuya postcondición es q.

$wp(\texttt{x:=x*x; y:= y+1}, x < y) =$
$wp(\texttt{x:=x*x}, wp(\texttt{y:= y+1}, x < y) = wp(\texttt{x:=x*x}, x < y + 1) =$
$x^2 < y + 1$

Precondiciones más débiles

$$
\begin{aligned}
&wp(\texttt{x:=x+a; y:= y-1}, x = (b-y)\cdot a) = \\
&= wp(\texttt{x:=x+a}, wp(\texttt{y:= y-1}, x = (b-y)\cdot a) \\
&= wp(\texttt{x:=x+a}, x = (b-y+1)\cdot a) = x+a = (b-y+1)\cdot a \\
&\equiv x = (b-y)\cdot a
\end{aligned}
$$

Dada la precondición $x = (b-y)\cdot a$, la expresión del ejemplo, entendida como transformador de predicados, no hace nada. A este tipo de predicados se les llama invariantes.

Precondiciones más débiles

$$wp(\texttt{si B entonces S1 si no S2}, q) = \\ = (B \rightarrow wp(\texttt{S1}, q)) \wedge (\neg B \rightarrow wp(\texttt{S2}, q))$$

$$wp(\texttt{si B entonces S1 si no S2}, q) = \\ = (B \wedge wp(\texttt{S1}, q)) \vee (\neg B \wedge wp(\texttt{S2}, q))$$

La primera de las definiciones es fácil de entender ya que el predicado B crea una partición del conjunto de estados en dos conjuntos disjuntos y las precondiciones están determinadas por las acciones de cada `Si` en su subconjunto. La definición equivalente se sigue de la equivalencia lógica:

$$(p \rightarrow q) \wedge (\neg p \rightarrow r) \equiv (p \wedge q) \vee (\neg p \wedge r)$$

Precondiciones más débiles

$$
\begin{aligned}
&wp(\texttt{si y=0 entonces x:=0 si no x:= y+1}, x = y) = \\
&= ((y = 0) \rightarrow wp(\texttt{x:=0}, x = y)) \wedge (y \neq 0 \rightarrow wp(\texttt{x:= y+1}, x = y)) \\
&= ((y = 0) \rightarrow (y = 0)) \wedge (y \neq 0 \rightarrow (y + 1 = y)) \equiv T \wedge ((y \neq 0) \rightarrow F) \\
&\equiv \neg(y \neq 0) \\
&\equiv y = 0
\end{aligned}
$$

$$
\begin{aligned}
&wp(\texttt{si y=0 entonces x:=0 si no x:= y+1}, x = y) \\
&= ((y = 0) \wedge wp(\texttt{x:=0}, x = y)) \vee ((y \neq 0) \wedge wp(\texttt{x:= y+1}, x = y)) \\
&= ((y = 0) \wedge (y = 0)) \vee ((y \neq 0) \wedge (y + 1 = y)) \equiv (y = 0) \vee F \\
&\equiv (y = 0)
\end{aligned}
$$

Precondiciones más débiles

Las dos definiciones de condición más débil para expresiones `mientras` siguientes son equivalentes:

$$wp(\texttt{mientras B hacer S}, q) =$$
$$= (\neg B \rightarrow q) \wedge (B \rightarrow wp(\texttt{S; mientras B hacer S}, q))$$
$$wp(\texttt{mientras B hacer S, q}) =$$
$$= (\neg B \wedge q) \vee (B \wedge wp(\texttt{S; mientras B hacer S}, q))$$

La segunda definición se sigue de la primera aplicando la misma equivalencia que en la definición de condición más débil para las expresiones `Si`. En cuanto a la primera se entiende a la luz de la ejecución de una expresión `mientras`:

- La expresión termina inmediatamente si la expresión booleana se evalúa falsa, en cuyo caso el estado no cambia por lo que precondición y postcondición son la misma.
- La expresión booleana se evalúa como cierta y causa la ejecución del cuerpo del bucle. Hasta el final de la ejecución la expresión `mientras` trata de establecer la postcondición.

Precondiciones más débiles

Dada la definición recursiva de la condición más débil para una expresión `mientras`, ésta no se puede establecer de manera constructiva. Sin embargo, veremos un ejemplo que ayuda a entender la situación. Llamemos W a

```
mientras x>0 hacer x:=x-1.
```

En ese caso,

$$
\begin{aligned}
& wp(W, x = 0) \\
&= (\neg(x > 0) \wedge (x = 0)) \vee ((x > 0) \wedge wp(\texttt{x:= x-1; } W, x = 0)) \\
&\equiv (x = 0) \vee ((x > 0) \wedge wp(\texttt{x:= x-1; } W, x = 0)) \\
&\equiv (x = 0) \vee ((x > 0) \wedge wp(W, x = 0)\{x \leftarrow x - 1\}).
\end{aligned}
$$

Precondiciones más débiles

Ahora debemos realizar la sustitución $\{x \leftarrow x-1\}$ en $wp(W, x=0)$.
Haciendo la sustitución y simplificando, se tiene:

$$
\begin{aligned}
& wp(W, x=0) \\
& = (x=0) \vee (x=1) \vee ((x>0) \wedge \\
& \wedge wp(W, x=0)\{x \leftarrow x-1\}\{x \leftarrow x-1\}).
\end{aligned}
$$

Continuando con la computación, se llega a:

$$
\begin{aligned}
& wp(W, x=0) \\
& = (x=0) \vee (x=1) \vee (x=2) \vee \ldots \\
& \equiv x \geq 0
\end{aligned}
$$