LENGUAJES Y COMPILADORES

Lógica de predicados

Extendemos nuestro mini-lenguaje de la siguiente manera:

<intexp>::= 0 | 1 | 2 | ...
| <var>
| -<intexp>
| <intexp> + <intexp>| <intexp> - <intexp>
| <intexp> * <intexp>| <intexp> ÷ <intexp>| <intexp> % <intexp>
<assert>::= **true**| **false**
| <intexp> = <intexp>| <intexp> ≠ <intexp>| <intexp>≤ <intexp>
| <intexp> ≥ <intexp>| <intexp> <<intexp>| <intexp>><intexp>
| ¬ <assert>
| <assert>∧ <assert>| <assert>∨ <assert>
| <assert>⇒ <assert>| <assert>⇔ <assert>
| ∀ <var>. <assert>| ∃ <var>. <assert>

Dominios semánticos. Necesitamos ahora dos dominios semánticos: uno para dar significado a las frases enteras y otro para dar significado a las frases booleanas. Extendiendo lo realizado para el mini-lenguaje de la manera obvia tenemos $\mathbb{Z}$ para dar significado a las frases enteras y $\mathbb{B} = \{V, F\}$ para dar significado a las frases booleanas.

Pero estos dominios no son suficiente ahora que tenemos variables. ¿Cuál sería el significado de x+3? ¿o el de x >5+y? ¿V o F? Depende del valor de la variables. Diremos que depende del "estado", donde el estado es justamente una función que asigna valores a las variables:

$$\Sigma = \text{<var>} \to \mathbb{Z}$$

El conjunto Σ es el de todos los estados posibles. Ahora sí, podemos decir que el significado de una expresión entera e será una función que dependiendo del estado σ dará un entero que es el valor de e en σ; es decir, una función de $\Sigma \to \mathbb{Z}$. Similarmente, el significado de una frase booleana será un función de $\Sigma \to \mathbb{B}$.

$$\begin{aligned}
[\![\ ]\!] &\in \Sigma \to \mathbb{Z} \\
[\![0]\!]\sigma &= 0 \\
[\![-e]\!]\sigma &= -[\![e]\!]\sigma \\
[\![e_0 + e_1]\!]\sigma &= [\![e_0]\!]\sigma + [\![e_1]\!]\sigma \\
[\![e_0 - e_1]\!]\sigma &= [\![e_0]\!]\sigma - [\![e_1]\!]\sigma \\
[\![e_0 * e_1]\!]\sigma &= [\![e_0]\!]\sigma * [\![e_1]\!]\sigma \\
[\![e_0 \div e_1]\!]\sigma &= [\![e_0]\!]\sigma \div [\![e_1]\!]\sigma \\
[\![e_0 \,\%e_1]\!]\sigma &= [\![e_0]\!]\sigma \,\%[\![e_1]\!]\sigma \\
[\![v]\!]\sigma &= \sigma\ v
\end{aligned}$$

Antes de dar las ecuaciones semánticas correspondientes a las expresiones booleanas, definimos la siguiente operación sobre estados. Sea $\sigma \in \Sigma$ un estado, $v \in$ <var> y $n \in \mathbb{Z}$. Entonces $[\sigma|v : n]$ es un estado que coincide con σ en todas las variables salvo posiblemente en v, donde este nuevo estado tiene asignado n. En símbolos:

$$[\sigma|v : n]\ w = \begin{cases} n & \text{si } w = v \\ \sigma\ w & \text{si no} \end{cases}$$

Esta misma notación se usa a lo largo de la materia cada vez que se quiere modificar una función en uno solo de sus posibles argumentos. Además, se puede iterar, por ejemplo $[[\sigma|v_0 : n_0]|v_1 : n_1]$ que se abrevia $[\sigma|v_0 : n_0|v_1 : n_1]$. La abreviatura se generaliza obteniendo $[\sigma|v_0 : n_0|v_1 : n1| \ldots |v_{k-1} : n_{k-1}]$.

Pregunta. ¿Qué ocurre cuando hay repetición de variables? Por ejemplo, sea el estado $\sigma' = [\sigma|v_0 : n_0|v_1 : n1| \ldots |v_{k-1} : n_{k-1}]$. Si $v_i = v_j$ (con $i < j$), cuánto es $\sigma'\ v_i$?

Por ejemplo, si tomamos la función coseno cos, la función $[\cos|\pi/2 : 1|\pi : 2|3\pi/2 : 1]$ es idéntica a la función cos salvo en los puntos $\pi/2$, π y $3\pi/2$. También observar lo que ocurre si modificamos esta última función

$$\begin{array}{rcl} [[\cos|\pi/2 : 1|\pi : 2|3\pi/2 : 1]|\pi/2 : 0] & = & [\cos|\pi/2 : 1|\pi : 2|3\pi/2 : 1|\pi/2 : 0] \\ & = & [\cos|\pi/2 : 0|\pi : 2|3\pi/2 : 1] \\ & = & [\cos|\pi : 2|3\pi/2 : 1] \end{array}$$

Ahora sí, las ecuaciones semánticas correspondientes a las expresiones booleanas:

$$\begin{array}{rcl} \{\!\{\ \}\!\} & \in & <\text{assert}> \to \mathbb{B} \\ \{\!\{\mathbf{true}\}\!\}\sigma & = & V \\ \{\!\{\mathbf{false}\}\!\}\sigma & = & F \\ \{\!\{e_0 = e_1\}\!\}\sigma & = & \begin{cases} V & \text{si } [\![e_0]\!]\sigma = [\![e_1]\!]\sigma \\ F & \text{si no} \end{cases} \\ & = & [\![e_0]\!]\sigma = [\![e_1]\!]\sigma \\ \{\!\{e_0 \neq e_1\}\!\}\sigma & = & [\![e_0]\!]\sigma \neq [\![e_1]\!]\sigma \\ \{\!\{e_0 \leq e_1\}\!\}\sigma & = & [\![e_0]\!]\sigma \leq [\![e_1]\!]\sigma \\ \{\!\{e_0 \geq e_1\}\!\}\sigma & = & [\![e_0]\!]\sigma \geq [\![e_1]\!]\sigma \\ \{\!\{e_0 < e_1\}\!\}\sigma & = & [\![e_0]\!]\sigma < [\![e_1]\!]\sigma \\ \{\!\{e_0 > e_1\}\!\}\sigma & = & [\![e_0]\!]\sigma > [\![e_1]\!]\sigma \\ \{\!\{e_0 > e_1\}\!\}\sigma & = & [\![e_0]\!]\sigma > [\![e_1]\!]\sigma \\ \{\!\{\neg b\}\!\}\sigma & = & \neg\{\!\{b\}\!\}\sigma \\ \{\!\{b_0 \wedge b_1\}\!\}\sigma & = & \{\!\{b_0\}\!\}\sigma \wedge \{\!\{b_1\}\!\}\sigma \\ \{\!\{b_0 \vee b_1\}\!\}\sigma & = & \{\!\{b_0\}\!\}\sigma \vee \{\!\{b_1\}\!\}\sigma \\ \{\!\{b_0 \Rightarrow b_1\}\!\}\sigma & = & \{\!\{b_0\}\!\}\sigma \Rightarrow \{\!\{b_1\}\!\}\sigma \\ \{\!\{b_0 \Leftrightarrow b_1\}\!\}\sigma & = & \{\!\{b_0\}\!\}\sigma \Leftrightarrow \{\!\{b_1\}\!\}\sigma \\ \{\!\{\forall v.b\}\!\}\sigma & = & \begin{cases} V & \text{si para todo } n \in \mathbb{Z},\ \{\!\{b\}\!\}[\sigma|v : n] = V \\ F & \text{si no} \end{cases} \\ & = & \forall n \in \mathbb{Z}.\{\!\{b\}\!\}[\sigma|v : n] \\ \{\!\{\exists v.b\}\!\}\sigma & = & \exists n \in \mathbb{Z}.\{\!\{b\}\!\}[\sigma|v : n] \end{array}$$

Observaciones. Son dos funciones, $[\![\ ]\!]$ y $\{\!\{\ \}\!\}$ pero usaremos indistintamente la notación $[\![\ ]\!]$ para ambas.

Como puede verse, la metacircularidad abunda en esta definición. Acá se da el problema que mencionamos más arriba, al hablar de metacircularidad en el mini-lenguaje: sin darnos cuenta hemos copiado al lenguaje, vicios del metalenguaje: los problemas con la división. Por ahora, y por un buen rato, asumiremos que la división está siempre definida, incluso si el divisor es 0.

Es importante observar que la definición es dirigida por sintaxis. Entonces definen el significado de manera única y la semántica resultante es composicional.

Ejemplos.

$$\begin{aligned}
[\![\mathbf{true}]\!]\sigma &= V \\
[\![x + 0 = x]\!]\sigma &= ([\![x + 0]\!]\sigma = [\![x]\!]\sigma) \\
&= ([\![x]\!]\sigma + [\![0]\!]\sigma = \sigma\ x) \\
&= (\sigma\ x + 0 = \sigma x) \\
&= (\sigma\ x = \sigma x) \\
&= V \\
[\![\forall x.x + 0 = x]\!]\sigma &= \forall n \in \mathbb{Z}.[\![x + 0 = x]\!][\sigma|x : n] \\
&= \forall n \in \mathbb{Z}.V \\
&= V
\end{aligned}$$

Variables y metavariables. Ahora que tenemos variables en el lenguaje, es oportuno repasar la noción de metavariable: es una variable del metalenguaje. En las ecuaciones que definen la semántica, ejemplos de metavariables son e, e_0 y e_1 (corren sobre expresiones enteras), b, b_0 y b_1 (corren sobre expresiones booleanas), n (corre sobre enteros) y σ (corre sobre estados). Incluso v no es una variable del lenguaje. Es en realidad una metavariable que corre sobre las variables del lenguaje. Si v fuera una variable del lenguaje, deberíamos hacer otra ecuación para cada una de las otras variables del lenguaje. En vez de eso, se hace una sola ecuación donde v representa a una variable cualquiera. Esto la torna una metavariable. Usaremos u, v, w para metavariables y x, y, z para variables del lenguaje. En los ejemplos de arriba, x es *la* variable x.

Por ejemplo, en la frase $\forall x.\exists y.y > x$, x e y son *las* variables x e y. Por lo tanto, sabemos que son variables diferentes. En cambio, en $\forall v.\exists w.w > v$, tratándose de metavariables no está excluida la posibilidad de que v y w sean iguales, (por ejemplo, que ambas sean *la* variable z).

Ligadura. Además de composicionalidad, hay otras propiedades deseables que la semántica debería satisfacer. Por ejemplo, la semántica de una frase no debería depender del nombre de sus variables ligadas, es decir, $\forall x.x + 0 = x$ debería tener el mismo significado que $\forall y.y + 0 = y$.

Para formular ésta y otras propiedades es necesario introducir las siguientes definiciones:

Ocurrencia ligadora: una ocurrencia ligadora de una variable es la que se encuentra inmediatamente después de un cuantificador ($\forall$ o $\exists$).

Alcance de una ocurrencia ligadora: En $\forall v.p$ o $\exists v.p$, p es el alcance de la ocurrencia ligadora de v que se encuentra antes de p.

Ocurrencia ligada: cualquier ocurrencia de v en el alcance de una ocurrencia ligadora de v es una ocurrencia ligada de v. Dicha ocurrencia ligada puede estar en más de un alcance de ocurrencias ligadoras de v. En ese caso, se dice que está ligada por la ocurrencia ligadora de menor alcance.

Ocurrencia libre: una ocurrencia de una variable que no es ligadora ni ligada es una ocurrencia libre.

Variable libre: : una variable que tiene ocurrencias libres en p es una variable libre de p.

Expresión cerrada: : es una expresión que no tiene variables libres.

Ejemplo. En la frase booleana

$$\begin{array}{cccccccccc} \forall\ x\ .\ (x & = & y & \wedge & \forall\ y\ .\ (x & = & y & \wedge & \exists\ x\ .\ x & + & y & = & x)) \\ 0 & 1 & 2 & & 3 & 4 & 5 & & 6 & 7 & 8 & 9 \end{array}$$

tenemos lo siguiente:

- ocurrencias ligadoras: 0, 3 y 6.
- ocurrencias ligadas: 1 y 4 (ligadas por 0), 5 y 8 (ligadas por 3) y 7 y 9 (ligadas por 6). Observar que 7 y 9 también están en el alcance de 0, pero el alcance de 6 es menor.
- ocurrencias libres: solo la ocurrencia 2.
- variables libres: sólo y es libre.

Observar que x e y son variables libres de $x + y = x$, pero sólo y es variable libre de $\exists\ x\ .\ x+y = x$. Si miramos ahora la expresión más grande $\forall\ y\ .\ (x = y \wedge \exists\ x\ .\ x+y = x)$, la variable y dejó de ser libre, ahora la variable x es libre.

Variables libres. Se puede definir el conjunto de variables libres por ecuaciones dirigidas por sintaxis. Para expresiones enteras ($\mathcal{P}_f$ devuelve el conjunto de subconjuntos finitos):

$$\begin{array}{rcl} FV & \in & \text{<intexp>} \to \mathcal{P}_f(\text{<var>}) \\ FV(0) & = & \{\} \\ FV(v) & = & \{v\} \\ FV(-e) & = & FV(e) \\ FV(e_0 + e_1) & = & FV(e_0) \cup FV(e_1) \\ & \vdots & \end{array}$$

y para expresiones booleanas

$$\begin{array}{rcl} FV & \in & \text{<assert>} \to \mathcal{P}_f(\text{<var>}) \\ FV(\textbf{true}) & = & \{\} \\ FV(\textbf{false}) & = & \{\} \\ FV(e_0 = e_1) & = & FV(e_0) \cup FV(e_1) \\ & \vdots & \\ FV(\neg b) & = & FV(b) \\ FV(b_0 \wedge b_1) & = & FV(b_0) \cup FV(b_1) \\ & \vdots & \\ FV(\forall v.b) & = & FV(b) - \{v\} \\ FV(\exists v.b) & = & FV(b) - \{v\} \end{array}$$

Calculando para el ejemplo anterior $p = \forall x.(x = y \wedge \forall y.(x = y \wedge \exists x.x + y = x))$

$$\begin{array}{rcl} FV(p) & = & FV(\forall x.(x = y \wedge \forall y.(x = y \wedge \exists x.x + y = x))) \\ & = & FV(x = y \wedge \forall y.(x = y \wedge \exists x.x + y = x)) - \{x\} \\ & = & (FV(x = y) \cup FV(\forall y.(x = y \wedge \exists x.x + y = x))) - \{x\} \\ & = & (\{x, y\} \cup (FV(x = y \wedge \exists x.x + y = x) - \{y\})) - \{x\} \\ & = & (\{x, y\} \cup ((FV(x = y) \cup FV(\exists x.x + y = x)) - \{y\})) - \{x\} \\ & = & (\{x, y\} \cup ((\{x, y\} \cup (FV(x + y = x) - \{x\})) - \{y\})) - \{x\} \\ & = & (\{x, y\} \cup ((\{x, y\} \cup (\{x, y\} - \{x\})) - \{y\})) - \{x\} \\ & = & (\{x, y\} \cup ((\{x, y\} \cup \{y\}) - \{y\})) - \{x\} \\ & = & (\{x, y\} \cup (\{x, y\} - \{y\})) - \{x\} \\ & = & (\{x, y\} \cup \{x\}) - \{x\} \\ & = & \{x, y\} - \{x\} \\ & = & \{y\} \end{array}$$

Las ecuaciones son dirigidas por sintaxis, esto garantiza que define unívocamente la función FV.

Sustituciones. Las expresiones con variables libres pueden instanciarse sustituyendo sus variables libres por términos. A continuación, definimos la operación de sustitución. Sea

$$\Delta = <\text{var}> \rightarrow <\text{intexp}>$$

el conjunto de todas las sustituciones. Una sustitución, es una función de variables en expresiones enteras (dado que el lenguaje de la lógica de predicados sólo tiene variables enteras, no hay variables lógicas).

Para expresiones enteras:

$$\begin{array}{rcl} \_/\_ & \in & <\text{intexp}> \times \Delta \rightarrow <\text{intexp}> \\ 0/\delta & = & 0 \\ v/\delta & = & \delta\ v \\ (-e)/\delta & = & -(e/\delta) \\ (e_0 + e_1)/\delta & = & (e_0/\delta) + (e_1/\delta) \\ & \vdots & \end{array}$$

Intuitivamente podemos pensar que una sustitución se propaga por toda la estructura de la expresión entera salvo cuando se encuentra con una variable, en cuyo caso reemplaza según indica la sustitución. Es que las expresiones enteras no tienen cuantificadores, por ello todas las variables de una expresión entera e son libres (aunque pueden no ser variables libres de una expresión booleana mayor que contiene a e como subexpresión) y por ello aplicar una sustitución es sencillo.

Por ejemplo, sea $id \in <\text{var}> \rightarrow <\text{intexp}>$ la sustitución identidad (sustituye la variable v por la expresión entera v) y $\delta = /[id|x : e_0|y : e_1|z : e_2]$

$$\begin{array}{rcl} (x + y * z - 5)/\delta & = & (x/\delta) + (y/\delta) * (z/\delta) - (5/\delta) \\ & = & \delta\ x + (\delta\ y) * (\delta\ z) - 5 \\ & = & e_0 + e_1 * e_2 - 5 \end{array}$$

Observar que en las ecuaciones que definen la aplicación de una sustitución sólo hay sintaxis. En la ecuación $(-e)/\delta = -(e/\delta)$ el signo "-" a ambos lados de la ecuación denota lo mismo: el operador "-" unario. Ninguno de esos operadores es la función opuesto del metalenguaje. De todas formas sigue habiendo metavariables (v, e, e_0, e_1, δ).

La sustitución para expresiones booleanas se define como sigue:

$$\begin{array}{rcl} \_/\_ & \in & <\text{assert}> \times \Delta \to <\text{assert}> \\ \textbf{true}/\delta & = & \textbf{true} \\ \textbf{false}/\delta & = & \textbf{false} \\ (e_0 = e_1)/\delta & = & (e_0)/\delta = (e_1)/\delta \\ & \vdots & \\ (\neg b)/\delta & = & \neg(b/\delta) \\ (b_0 \wedge b_1)/\delta & = & (b_0/\delta) \wedge (b_1/\delta) \\ & \vdots & \end{array}$$

Hasta acá, es totalmente análogo al caso de expresiones enteras y valen las mismas reflexiones que hiciéramos ahí.

El problema de la captura. Cuando consideramos cuantificadores aparece una dificultad. Por ejemplo, supongamos que queremos aplicar una sustitución δ a la frase $\exists x \,.\, x > y$, que es intuitivamente válida ya que no importa el valor de y, siempre existe un entero mayor x. Una aplicación "ingenua" de la sustitución, me daría $\exists x \,.\, x > e$ donde $e = \delta\ y$. En efecto, la variable x no se toca porque está ligada. Pero ¿qué pasa si e es x?, quedaría $\exists x \,.\, x > x$ que es falsa. Lo mismo ocurre, por ejemplo, si $e = x + 1$, quedaría $\exists x \,.\, x > x + 1$ que también es falsa.

Otro problema que puede observarse, es que como $\exists x \,.\, x > y$ por un lado, y $\exists z \,.\, z > y$ por otro sólo se diferencian en el nombre de la variable ligada, sustituir en uno u otro me debería dar resultados equivalentes. Pero, nuevamente para el caso en que $\delta\ y = x + 1$, en un caso obtenemos $\exists x \,.\, x > x + 1$, y en el otro $\exists z \,.\, z > x + 1$. No son equivalentes (la primera es falsa y la segunda es válida).

El problema es que al sustituir δ en $\exists x \,.\, x > y$, se está **capturando** la ocurrencia de x que era libre en $x + 1$, ahora pasa a ser ligada. Eso no debería ocurrir. Una solución es renombrar la variable ligada obteniendo por ejemplo $\exists z \,.\, z > y$ y luego sustituir, obteniendo $\exists z \,.\, z > x + 1$.

Otra posibilidad es hacer las dos cosas simultáneamente. Si bien es más complicado, es lógico hacerlo así ya que estamos definiendo justamente la sustitución, no deberíamos asumir que sabemos renombrar, ya que renombrar es sustituir un nombre por otro (en realidad esto es discutible, ya que el renombre es una forma un poco más sencilla de sustitución).

Conclusión: al aplicar una sustitución a una expresión de la forma $\forall v \,.\, b$ o $\exists v \,.\, b$, se propaga la sustitución y simultáneamente se renombra la variable v:

$$(\forall v \,.\, b)/\delta = \forall u \,.\, (b/[\delta | v : u])$$

donde u es una variable *nueva* y $[\delta | v : u]$ ya se definió: sustituye parecido a δ, salvo que a la variable v la reemplaza por u. Si no elegimos cuidadosamente a u, el problema de la captura persiste. Volviendo al ejemplo anterior, si en $\exists x \,.\, x > y$ aplicamos δ, donde

$\delta\ y = x + z$, si sustituimos sin renombrar queda $\exists x\ .\ x > x + z$ que captura la x. Si renombramos x por z, nos queda $\exists z\ .\ z > x + z$ que captura la z. Para evitar la captura hay que elegir bien la variable nueva u: la variable u no debe ser **capturable**. ¿Y cuáles son las variables çapturables¿ Son las que pueden aparecer por culpa de δ al aplicar la sustitución en b. Recordemos que δ sólo debe sustituir las variables libres de $\forall v\ .\ b$. O sea que las "capturables" son las que introduce δ para cada variable libre de $\forall v\ .\ b$. O sea:

$$u \notin \bigcup_{w \in FV(\forall v\ .\ b)} FV(\delta\ w)$$

o equivalentemente

$$u \notin \bigcup_{w \in FV(b) - \{v\}} FV(\delta\ w)$$

Para finalizar, entonces, agregamos las ecuaciones

$$\begin{array}{lcll} (\forall v\ .\ b)/\delta & = & \forall u\ .\ (b/[\delta|v:u]) & \text{donde } u \notin \bigcup_{w \in FV(\forall v\ .\ b)} FV(\delta\ w) \\ (\exists v\ .\ b)/\delta & = & \exists u\ .\ (b/[\delta|v:u]) & \text{donde } u \notin \bigcup_{w \in FV(\forall v\ .\ b)} FV(\delta\ w) \end{array}$$