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 2 dominios semánticos: uno para dar significado a las frases enteras y otro para dar significado a frases booleanas. Extendiendo lo realizado para el mini-lenguaje de la manera obvia tenemos Z para dar significado a las frases enteras y 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:

Σ = <var> -> Z

Σ es el conjunto de todos los estados posibles. Ahora sí, podemos decir que el significado de una expresión entera e será una función que dependiendo del estado s dará un entero que es el valor de e en s; es decir, una función de Σ en Z. Similarmente el significado de una frase booleana será una función de Σ en B:

```
[[_]] ∈ <intexp> -> Σ -> Z
[[0]]s = 0 ...
[[-e]]s = - [[e]]s
[[e+f]]s = [[e]]s + [[f]]s
[[e-f]]s = [[e]]s - [[f]]s
[[e*f]]s = [[e]]s * [[f]]s
[[e÷f]]s = [[e]]s ÷ [[f]]s
[[e%f]]s = [[e]]s % [[f]]s
[[v]]s = s v
```

Antes de dar las ecuaciones semánticas correspondientes a las expresiones booleanas, definimos la siguiente operación sobre estados. Sea s ∈ Σ un estado, v una variable y n un entero. Entonces [s|v:n] es un estado que coincide con s en todas las variables salvo posiblemente en v, donde este nuevo estado tiene asignado n. En símbolos:

```
[s|v:n] w = si w = v entonces n sino s w
```

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 [[s|v:n]|w:m] es el estado que cuando se aplica a w da m, cuando se aplica a v (suponiendo que v ≠ w) da n, y cuando se aplica a cualquier otra variable u da s u. Para simplificar se escribe [s|v1:n1|...|vk:nk] en lugar de [[...[s|v1:n1]|...]|vk:nk].

Por ejemplo, si f es una función real, la función [f|π:1|4:2|6:3] es idéntica a la función f salvo en los puntos π, 4 y 6. Observar lo que ocurre si modifico esta última función en 4, recomponiendo su valor original f(4):

```
[[f|π:1|4:2|6:3] | 4:f(4)] = [f|π:1|4:2|6:3|4:f(4)]
                           = [f|π:1|4:f(4)|6:3]
                           = [f|π:1|6:3]
```

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

```
{{_}} ∈ <assert> -> Σ -> B
{{true}}s = V
{{false}}s = F
{{e = f}}s = si [[e]]s es igual a [[f]]s entonces V sino F
           = [[e]]s = [[f]]s
{{e ≠ f}}s = [[e]]s ≠ [[f]]s
{{e ≤ f}}s = [[e]]s ≤ [[f]]s
{{e ≥ f}}s = [[e]]s ≥ [[f]]s
{{e < f}}s = [[e]]s < [[f]]s
{{e > f}}s = [[e]]s > [[f]]s
{{¬b}}s = ¬ {{b}}s
{{b ∧ c}}s = {{b}}s ∧ {{c}}s
{{b ∨ c}}s = {{b}}s ∨ {{c}}s
{{b ⇒ c}}s = {{b}}s ⇒ {{c}}s
{{b ⇔ c}}s = {{b}}s ⇔ {{c}}s
{{∀v. b}}s = si para todo n en Z, {{b}}[s|v:n] es V, entonces V, sino F
           = ∀n∈Z. {{b}}[s|v:n]
{{∃v. b}}s = si para algún n en Z, {{b}}[s|v:n] es V, entonces V, sino F
           = ∃n∈Z. {{b}}[s|v:n]
```

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. Hasta dentro de unos 45 días, 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:

```
    [[true]]s = V
    [[x+0 = x]]s = ([[x+0]]s = [[x]s)
                 = ([[x]]s + [[0]]s = s x)
                 = (s x + 0 = s x)
                 = V
    [[∀x. x+0 = x]]s = ∀n∈Z. [[x+0 = x]] [s|x:n]
                     = ∀n∈Z. V
                     = V
```

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 y f (corren sobre expresiones enteras), b y c (corren sobre expresiones booleanas), n (corre sobre enteros) y s (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 c/u 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 ∀x. ∃y. y > x, los símbolos x e y son "las" variables x e y. Por lo tanto, sabemos que son variables diferentes. En cambio, en ∀v. ∃w. w > v, tratándose de metavariables no está excluida la posibilidad de que v y w sean iguales, (por ejemplo ambas iguales a z).

LIGADURA

El lenguaje de la lógica, como muchos otros lenguajes expresivos, incorporan la posibilidad de que una variable jueguen dos roles claramente diferenciados: la variable puede referirse a un objeto no específico, genérico dentro de un universo, o puede remitir a un objeto particular. Para permitir este fenómeno en la lógica de predicados se diferencia entre variables ligadas (o acotadas) y variables libres.

Además de composicionalidad, hay otras propiedades deseables que la semántica debería satisfacer. Por ejemplo, el significado de una frase no debería depender del nombre de las variables ligadas, es decir,

∀x. x+0 = x debería tener el mismo significado que ∀y. y+0 = y.

Para formular dichas 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 (∀ o ∃).
Alcance de una ocurrencia ligadora: En ∀v. p o ∃v. p, p es el alcance de la ocurrencia ligadora de v.
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. 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 es una variable libre.
Una expresión cerrada es una que no tiene variables libres.

Ejemplo: en

```
∀ x . (x = y ∧ ∀ y . (x = y ∧ ∃ x . x + y = x))
   0    1  2       3    4   5      6   7   8   9
```

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.
courrencias libres: 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 ∃ x . x + y = x. Si miramos ahora la expresión más grande ∀ y . (x = y ∧ ∃ 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:

FV(0) = {} ...
FV(v) = {v}
FV(-e) = FV(e)
FV(e+f) = FV(e) ∪ FV (f) ...

Para expresiones booleanas:

FV(true) = FV(false) = {}
FV(e = f) = FV(e) ∪ FV (f) ...
FV(¬b) = FV(b)
FV(b ∧ c) = FV(b) ∪ FV (c) ...
FV(∀v. b) = FV(b) - {v}
FV(∃v. b) = FV(b) - {v}

Calculando para el ejemplo anterior

```
FV(∀x. (x = y ∧ ∀y. (x = y ∧ ∃x. x + y = x))) = FV(x = y ∧ ∀y. (x = y ∧ ∃x. x + y = x)) - {x}
                                              = (FV(x = y) ∪ FV(∀y. (x = y ∧ ∃x. x + y = x))) - {x}
                                              = ({x, y} ∪ (FV(x = y ∧ ∃x. x + y = x) - {y})) - {x}
                                              = ({x, y} ∪ (({x,y} ∪ FV(∃x. x + y = x)) - {y})) - {x}
                                              = ({x, y} ∪ (({x,y} ∪ (FV(x + y = x) - {x})) - {y})) -
{x}
                                              = ({x, y} ∪ (({x,y} ∪ ({x,y} - {x})) - {y})) - {x}
                                              = ({x, y} ∪ (({x,y} ∪ {y}) - {y})) - {x}
                                              = ({x, y} ∪ ({x,y} - {y})) - {x}
                                              = ({x, y} ∪ {x}) - {x}
                                              = {x, y} - {x}
                                              = {y}
```

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

SUSTITUCIONES

Las expresiones con variables libres pueden instanciarse sustituyendo sus variables libres por términos. A continuación definimos la operación de sustitución. Comenzamos definiendo el mapa sustitución como una función de variables en expresiones. Sea

Δ = <var> -> <intexp>

el conjunto de todos los mapas sustituciones (notar que el lenguaje de la lógica de predicados sólo tiene variables enteras, no hay variables lógicas).

Definimos ahora el operador sustitución, que "opera" sobre expresiones enteras (términos) y expresiones booleanas (predicados):

Para expresiones enteras:

_/_ ∈ <intexp> x Δ -> <intexp>
0/d = 0 ...
v/d = d v
(-e)/d = -(e/d)
(e+f)/d = (e/d)+(f/d) ...

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: si d x = e, d y = f, d z = g
(x + y*z - 5)/d = (x/d) + ((y*z - 5)/d)
                = e + ((y*z)/d) - (5/d)
                = e + (y/d)*(z/d) - 5
                = e + d y * d z - 5
                = e + f*g - 5

Observar que en las ecuaciones que definen la aplicación de una sustitución sólo hay sintaxis. En la ecuación (-e)/d = -(e/d) 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, d, e, f).

Para expresiones booleanas:

_/_ ∈ <assert> x Δ -> <assert>
true/d = true
false/d = false
(e=f)/d = (e/d)=(f/d) ...
(¬b)/d = ¬(b/d)
(b ∧ c)/d = (b/d) ∧ (c/d) ...

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