VARIABLES LIBRES Y VARIABLES ASIGNABLES

FV(skip) = {}
FV(v:= e) = {v} ∪ FV(e)
FV(c0;c1) = FV(c0) ∪ FV(c1)
FV(if b then c0 else c1) = FV(b) ∪ FV(c0) ∪ FV(c1)
FV(while b do c) = FV(b) ∪ FV(c)
FV(newvar v:= e in c) = FV(e) ∪ (FV(c) - {v})

FA(skip) = {}
FA(v:= e) = {v}
FA(c0;c1) = FA(c0) ∪ FA(c1)
FA(if b then c0 else c1) = FA(c0) ∪ FA(c1)
FA(while b do c) = FA(c)
FA(newvar v:= e in c) = FA(c) - {v}

Claramente FA(c) ⊆ FV(c), como puede comprobarse ecuación por ecuación.

TEOREMA DE COINCIDENCIA (TC)

¿ "si dos estados s y s' coinciden en las variables libres de c, entonces da lo mismo evaluar c en s o s'"
   (∀w∈FV(c). s w = s' w) => [[c]]s = [[c]]s' ?

Esto no vale si c es un comando, ya que devuelven estados finales que pueden diferir en las w ∉ FV(c) (porque tales diferencias podían existir previamente). Además, podría ocurrir que uno de ellos no termine, en cuyo caso ni siquiera devolvería un estado. Puede verse que si uno de ellos se cuelga, el otro también.

Teorema de Coincidencia (TC):
a) (∀w∈FV(c). s w = s' w) => o bien [[c]]s = ⊥ = [[c]]s', o bien [[c]]s ≠ ⊥ ≠ [[c]]s' y ∀w∈FV(c). [[c]]s w = [[c]]s' w
b) si [[c]]s ≠ ⊥, entonces ∀w∉FA(c). [[c]]s w = s w.

SUSTITUCIONES

Por culpa de la asignación v:= e, no podemos reemplazar una variable por una expresión entera arbitraria en un comando. Sólo podemos reemplazarla por otra variable:

Δ = <var> -> <var>

es ahora el conjunto de sustituciones. A pesar de este cambio, las ecuaciones que la definían para la lógica de predicados siguen valiendo (salvo el caso de los cuantificadores, que ahora no tenemos). Ahora se agregan ecuaciones para los comandos.

_/_ ∈ <comm> x Δ -> <comm>
skip/d = skip
(v:= e)/d = (d v):= (e/d)
(c0;c1)/d = (c0/d) ; (c1/d)
(if b then c0 else c1)/d = if b/d then c0/d else (c1/d)
(while b do c)/d = while b/d then (c/d)

El caso del newvar tiene los mismos inconvenientes que ∀ y ∃, ahora es un poco más simle ya que d w son siempre variables.

(newvar v:= e in c)/d = newvar v':= (e/d) in (c/[d|v:(e/d)])
donde v' ∉ {d w | w ∈ FV(c) - {v}}

TEOREMA DE SUSTITUCION (TS)

¿ "si aplico la sustitución d a c y luego evalúo en el estado s, puedo obtener el mismo resultado a partir de c sin sustituir si evalúo en un estado que hace el trabajo de d y de s (en las variables libres de c)"
   (∀w∈FV(c). [[d w]]s = s' w) => [[c/d]]s = [[c]]s' ?

Por los problemas mencionados para el TC, debería reescribirse así:
   (∀w∈FV(c). [[d w]]s = s' w) => o bien [[c/d]]s = ⊥ = [[c]]s', o bien [[c/d]]s ≠ ⊥ ≠ [[c]]s' y ∀w∈FV(c). [[c/d]]s (d w) = [[c]]s' w

Pero ni siquiera así vale TS. El problema ocurre cuando d manda dos variables a una misma variable. Por ejemplo, en el caso del programa

x:= x-1; y:= 2*y

podemos comprobar que para todo s' ∈ Σ,

[[x:= x-1; y:= 2*y]]s' = [s' | x : s' x - 1 | y : 2 * s' y]
                      = [[y:= 2*y; x:= x-1]]s'

Pero veamos qué pasa si tenemos la mala suerte de reemplazar x e y por la misma variable: d = x -> z, y -> z

```
[[(x:= x-1; y:= 2*y)/d]]s = [[z:= z-1; z:= 2*z]]s
                          = [s | z : 2 * (s z - 1)]
                          ≠ [s | z : 2 * s z - 1]
                          = [[z:= 2*z; z:= z-1]]s
                          = [[(y:= 2*y; x:= x-1)/d]]s
```

No es posible encontrar un s' ∈ Σ que por sólo hacer el trabajo de d y de s en las variables libres {x, y} obtenga [[(x:= x-1; y:= 2*y)/d]]s = [[x:= x-1; y:= 2*y]]s'
ya que con igual criterio obtendría también
[[(y:= 2*y; x:= x-1)/d]]s = [[y:= 2*y; x:= x-1]]s'
y esto es imposible ya que acabamos de demostrar que las partes derechas son iguales entre sí, pero las izquierdas son distintas entre sí.

El problema surge porque d no es inyectiva.

Ahora sí:

Teorema de Sustitución (TS):
Si d es inyectiva sobre FV(c) y (∀w∈FV(c). [[d w]]s = s' w), entonces o bien [[c/d]]s = ⊥ = [[c]]s', o bien [[c/d]]s ≠ ⊥ ≠ [[c]]s' y ∀w∈FV(c). [[c/d]]s (d w) = [[c]]s' w.

La demostración requiere de una generalización:
Lema de Sustitución (LS):
Sea FV(c) ⊆ V ⊆ <var> tal que d es inyectiva sobre V y (∀w∈V. [[d w]]s = s' w). Entonces, o bien [[c/d]]s = ⊥ = [[c]]s', o bien [[c/d]]s ≠ ⊥ ≠ [[c]]s' y ∀w∈V. [[c/d]]s (d w) = [[c]]s' w.

TEOREMA DE RENOMBRE (TN):
"no importan los nombres de las variables ligadas"
   u ∉ FV(c)-{v} => [[newvar u:= e in (c/v->u)]] = [[newvar v:= e in c]]

FALLAS

Se agregan excepciones al lenguaje:

<comm> ::= fail | catchin <comm> with <comm>

Ahora el comportamiento de un comando puede presentar 3 aspectos:
1) da un estado final
2) aborta y da un estado final
3) no termina

Sea Σ' = Σ ∪ {abort}xΣ con el orden discreto.

[[]] ∈ <comm> -> Σ -> Σ'_⊥

En Σ'_⊥ seguimos teniendo un dominio llano.

```
[[skip]]s = s
[[fail]]s = <abort, s>
[[v:= e]]s = [s | v : [[e]]s]
                            ⌈ [[c0]]s         si [[b]]s
[[if b then c0 else c1]]s = ⌊
                            ⌊ [[c1]]s         c.c.
[[c0;c1]]s = [[c1]]_* ([[c0]]s)
```
donde, dada una f ∈ Σ -> Σ'_⊥, denotamos por f_* la siguiente extensión de f a Σ'_⊥. Así, f_* ∈ Σ'_⊥ -> Σ'_⊥ definida por:

```
        ⌈ f x        si x ∈ Σ
f_* x = ⌊
        ⌊ x          c.c.
```

```
[[catchin c0 with c1]]s = [[c1]]_♣ ([[c0]]s)
```
donde, dada una f ∈ Σ -> Σ'_⊥, denotamos por f_♣ la siguiente extensión de f a Σ'_⊥. Así, f_♣ ∈ Σ'_⊥ -> Σ'_⊥ definida por:

```
         ⌈ f s      si x = <abort, s>
f_♣ x = ⌊
         ⌊ x        c.c.
```

```
[[while b do c]] = sup'(F^i ⊥')

              ⌈ w_* ([[c]]s)     si [[b]]s
donde F w s = ⌊
              ⌊ s                c.c.
```

[[newvar v:= e in c]]s = (λ s'∈Σ. [s' | v : s v])_† ([[c]][s | v : [[e]]s])
donde, dada una f ∈ Σ -> Σ, denotamos por f_† la siguiente extensión de f a Σ'_⊥. Así, f_† ∈ Σ'_⊥ -> Σ'_⊥ definida por:

$$f_\dagger\ x = \begin{cases} \langle abort, f\ s\rangle & \text{si } x = \langle abort, s\rangle \\ f\ x & \text{si } x \in \Sigma \\ \bot & \text{si } x = \bot \end{cases}$$

Ejercicio: reformular el teorema de coincidencia y el de sustitución para que tenga en cuenta los posibles nuevos comportamientos.