

Un Modelo Matemático Para Semánticas Operacionales Estructurales

Mauro Jaskelioff

mjj@cs.nott.ac.uk

School of Computer Science



The University of
Nottingham

FCEIA'07

Semántica de Lenguajes de Programación

Necesitamos especificar la semántica de los lenguajes de programación para:

- ▶ Comunicar el diseño.
- ▶ Optimizar programas
- ▶ Operar sobre el lenguaje.
- ▶ En gral, razonar sobre las propiedades del lenguaje.

Estilos de Semántica

Hay distintas formas de dar semánticas a un lenguaje:

Semántica Denotacional :

Dado un modelo matemático adecuado M , dar una función $\llbracket - \rrbracket : \text{Programa} \rightarrow M$

Semántica Operacional :

Especificar las transiciones que disparará un programa al ser ejecutado por una máquina abstracta (cuya semántica se supone trivial).

Otras : Axiomática, Juegos, etc.

Cada estilo tiene sus ventajas y sus problemas.

Pregunta Fundamental en Semántica

¿Cuándo puedo afirmar que dos programas p_1 y p_2 son equivalentes?

Respuesta:

- ▶ En semántica denotacional, cuando $\llbracket p_1 \rrbracket =_M \llbracket p_2 \rrbracket$.
- ▶ En semántica operacional, cuando $p_1 \sim p_2$
Decimos que $p_1 \sim p_2$ si cada transición de p_1 es 'simulable' por una transición de p_2 y viceversa.
- ▶ La bisimulación permite abstraernos de la naturaleza de las *configuraciones* (estados de la máquina abstracta.) Sólo importa el *comportamiento observable* de los programas.

Semánticas Operacionales Estructurales (SOS)

- ▶ Las máquinas abstractas pueden tener transiciones no esenciales.
- ▶ Es difícil razonar con máquinas abstractas arbitrarias.
- ▶ Plotkin propone definir las transiciones usando reglas de inferencia que respeten la estructura de la sintaxis del lenguaje.
- ▶ Se puede razonar usando inducción estructural.

Ejemplo de SOS

Álgebra de procesos básica (BPA):

$$t ::= \text{nil} \mid a \mid t; t \mid t + t$$

$$\frac{}{\text{nil} \downarrow} \quad \frac{}{a \xrightarrow{a} \text{nil}}$$

$$\frac{t \xrightarrow{a} t'}{t; u \xrightarrow{a} t'; u} \quad \frac{t \downarrow \quad u \xrightarrow{a} u'}{t; u \xrightarrow{a} u'} \quad \frac{t \downarrow \quad u \downarrow}{t; u \downarrow}$$

$$\frac{t \xrightarrow{a} t'}{t + u \xrightarrow{a} t'} \quad \frac{u \xrightarrow{a} u'}{t + u \xrightarrow{a} u'} \quad \frac{t \downarrow}{t + u \downarrow} \quad \frac{u \downarrow}{t + u \downarrow}$$

- ▶ El lenguaje no es determinístico
- ▶ El comportamiento observable de un proceso p es realizar alguna acción $a \in A$ y comportarse como un proceso p' ($p \xrightarrow{a} p'$) o terminar ($p \downarrow$)

Con el enfoque estructurado, las semánticas operacionales se hacen populares ya que son:

- ▶ Intuitivas.
- ▶ Fácilmente aplicables a lenguajes concurrentes.
- ▶ Si uno es cuidadoso, describen directamente el comportamiento observable.

Además, el principio de inducción estructural es poderoso y relativamente fácil de aplicar.

¿Qué es exactamente una SOS?

¿Cuándo un montón de reglas definen una semántica operacional estructural?

Sabemos que las reglas

- ▶ deben respetar la estructura del lenguaje
- ▶ deben inducir un sistema de transiciones
- ▶ ¿algo más?

Es deseable que la noción de equivalencia asociada sea una congruencia.

Objetivos

- ▶ Presentar una teoría matemática de SOS
- ▶ Trabajar en la forma más general posible.



The University of
Nottingham

Formatos de Regla

Los formatos de reglas

- ▶ Limitan la sintaxis de las reglas
- ▶ Garantizan determinadas propiedades (ej: bisimulación es una congruencia)
- ▶ Son demasiados concretos:
 - Están dados para un comportamiento observable fijo,
 - pero son un buen punto de partida.

Un formato de regla que garantiza que la bisimulación es una congruencia es GSOS [BIM95]

Definition (Formato de Regla GSOS)

Sean A_i y B_i subconjuntos de A . Una regla GSOS es una regla con la siguiente forma

$$\frac{\left\{ x_i \xrightarrow{a} y_{ij}^a \right\}_{\substack{1 \leq i \leq n, a \in A_i \\ 1 \leq j \leq m_i^a}} \quad \left\{ x_i \not\xrightarrow{b} \right\}_{b \in B_i}^{1 \leq i \leq n}}{\sigma(x_1, \dots, x_n) \xrightarrow{c} t}$$

tal que x_i y y_{ij}^a son distintos, y esas son las únicas variables que ocurren en el término t .

Demasiada sintaxis!

GSOS es demasiado concreto, y sólo se aplica a sistemas de transición etiquetados.

Para hacerlo más general tenemos que:

- ▶ abstraer la sintaxis
- ▶ abstraer el comportamiento observable
- ▶ abstraer el concepto de sistemas de transición (o máquinas abstractas)
- ▶ abstraer el formato de regla

Necesitamos una herramienta para manejar estas abstracciones:

Teoría de Categorías, por supuesto!

Sintaxis

- ▶ Consideramos lenguajes sin variables ligadas.
- ▶ Cada operación del lenguaje queda entonces caracterizada por su aridad.
- ▶ Para el lenguaje BPA, la sintaxis queda definida por el functor

$$\Sigma X = 1 + A + X \times X + X \times X$$

- ▶ en Haskell:

```
data S a = Nil | Act A | Sec a a | Opt a a
instance Functor S where
  fmap _ Nil           = Nil
  fmap _ (Act a)       = Act a
  fmap f (Sec t u)     = Sec (f t) (f u)
  fmap f (Opt t u)     = Opt (f t) (f u)
```

Términos del lenguaje

Definition (Álgebra de términos)

Dada una signatura Σ y un conjunto de variables X , el *álgebra de términos* $T_{\Sigma}(X)$ se define inductivamente como:

$$\frac{x \in X}{(\text{Var } x) \in T_{\Sigma}(X)} \qquad \frac{f \in \Sigma(n) \quad t_1, \dots, t_n \in T_{\Sigma}(X)}{f(t_1, \dots, t_n) \in T_{\Sigma}(X)}$$

En Haskell:

```
data T f x = Var x
           | Con (f (T f x))

instance Functor f => Functor (T f) where
instance Functor f => Monad (T f) where
foldT :: Functor f => (a -> b) -> (f b -> b)
      -> T f a -> b
```

Coálgebras

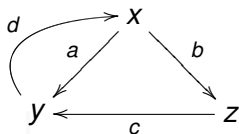
- ▶ Una coálgebra (en *Set*) una función $X \rightarrow BX$, donde X es el conjunto portador de la coálgebra y B un functor.
- ▶ Las coálgebras modelan sistema de transición, autómatas, máquinas abstractas, etc.
- ▶ X se puede pensar como un conjunto de estados.
- ▶ B representa el comportamiento observable.
- ▶ Partiendo de un estado $x \in X$, un sistema produce un cierto comportamiento observable y pasa a un nuevo estado (o más de un nuevo estado).

Ejemplo de coálgebra: LTS

Para $BX = \mathcal{P}(A \times X)$, una B -coálgebra representa un sistema de transición etiquetado.

Dado el conjunto de estados $X = \{x, y, z\}$, y un conjunto de acciones $A = \{a, b, c, d\}$

El sistema



está dado por la siguiente coálgebra

$$\begin{aligned}\alpha & : X \rightarrow \mathcal{P}(A \times X) \\ \alpha(x) & = \{(a, y), (b, z)\} \\ \alpha(y) & = \{(d, x)\} \\ \alpha(z) & = \{(c, y)\}\end{aligned}$$

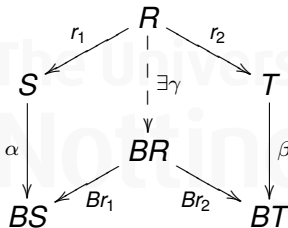
Noción de Equivalencia

Las coálgebras tienen una noción de equivalencia canónica:

Bisimulación B -coalgebraica

Para $s \in S, t \in T, R \subseteq S \times T$

$$\langle s, \alpha \rangle \sim_B \langle t, \beta \rangle \Leftrightarrow \exists \gamma$$



Reglas Operacionales Abstractas [TP97]

- ▶ Las coálgebras modelan sistemas de transición pero no SOS

Dados funtores

- ▶ Σ (signatura del lenguaje) y
- ▶ B (comportamiento observable)

Una regla operacional abstracta es una transformación natural

$$\Sigma \cdot (Id \times B) \Rightarrow B \cdot T_{\Sigma}$$

Prop: dada una regla abstracta operacional es posible obtener una coálgebra

$$T_{\Sigma}X \rightarrow BT_{\Sigma}X$$

Semántica de BPA

$$\Sigma X = 1 + A + X \times X + X \times X$$

Llamemos ι_{nil} , ι_a , $\iota_;$, ι_+ a las inyecciones en Σ

$$BX = \mathcal{P}(A \times X)$$

$$\rho: \Sigma(X \times BX) \rightarrow B(T_\Sigma X)$$

$$\rho_X(\iota_{\text{nil}} *) = \emptyset$$

$$\rho_X(\iota_a a) = \{(a, \text{nil})\}$$





$$\rho_X(\iota_((u, b_u), (v, b_v))) = \begin{cases} \{(a, \iota_((\eta u', \eta v') | (a, u') \in b_u))\} & \text{si } b_u \neq \emptyset \\ b_v & \text{en otro caso} \end{cases}$$

$$\rho_X(\iota_+((u, b_u), (v, b_v))) = b_u \cup b_v$$

Beneficios de trabajar en forma abstracta

- ▶ Soporte para variables ligadas, si trabajamos en un categoría de 'presheaves' $\mathcal{C}^{\mathbb{F}}$ [FT01]
- ▶ Soporte para diferentes efectos laterales
- ▶ Modificando la categoría base se obtienen diferentes nociones de equivalencias [Tur97]
- ▶ Análisis de modularidad de SOS
- ▶ Relación con semánticas denotacionales
- ▶ Simple fórmula matemática para SOS
- ▶ Es posible hablar de ideas sin demasiada sintaxis!

Referencias

-  Bard Bloom, Sorin Istrail, and Albert R. Meyer.
Bisimulation can't be traced.
J. ACM, 42(1):232–268, 1995.
-  Marcelo Fiore and Daniele Turi.
Semantics of name and value passing.
In *Proc. 16th LICS Conf.*, pages 93–104. IEEE, Computer Society Press, 2001.
-  D. Turi and G.D. Plotkin.
Towards a mathematical operational semantics.
In *Proc. 12th LICS Conf.*, pages 280–291. IEEE, Computer Society Press, 1997.
-  Daniele Turi.
Categorical modelling of structural operational rules: Case studies.
In *Category Theory and Computer Science*, 1997.