Semántica Denotacional de Lenguajes con Datos Borrosos

Daniel Sánchez Alvarez y Antonio F. Gómez Skarmeta
Departamento de Informática, Inteligencia Artificial y Electrónica
Universidad de Murcia
E-mail: daniel@dif.um.es

Resumen

Se da la semántica denotacional de un lenguaje de programación que maneja datos borrosos. La introducción de bloques plantea problemas en cuenta a la transmisión, tanto del grado con el que se trabaja, como de las operaciones triangulares necesarias para la evaluación de los grados de los datos borroso. Planteamos algunas soluciones. Se le dota de la posibilidad de definir variables lingüísticas

1 Introducción

En [4] se propone un lenguaje imperativo no determinista muy simple, queremos ampliarlo desde dos puntos de vista: bien enriqueciendo su estructuración, bien considerando la necesidades del cálculo borrosos. Con respecto a la primera, sería interesante ampliarlo para:

1. Dotarlo de estructura de bloques
2. Dotarlo funciones y procedimientos

Con respecto a la segunda, es interesante:

1. Elegir la manera de representar los conjuntos borrosos y las operaciones definidas sobre ellos
2. Poder definir variables lingüísticas

2 Bloques y Abstracciones

En dicho lenguaje existen dos elementos necesarios para la evaluación de una sentencia:

- El índice borroso, un elemento de $[0,1]$, en adelante lo representaremos por **Ind_Fuz** y
- Una de las funciones s obtenidas al definir el lambda cálculo_b [3], es decir una T-norma o una S-conorma. En adelante representaremos por **T_op** las tres funciones necesarias para calcular los valores de pertenencia de la intersección, unión y complementación de conjuntos borrosos.

El primero se modificaba en las ramificaciones, como consecuencia del resultado de la evaluación booleana de la prueba, y nos podía servir para una especie de inferencia borrosa *(modus ponens generalizado)*. El segundo quedaba algo ambiguo, nunca se especificaba si era la misma T-norma o S-conorma, en nuestro ánimo consideramos que siempre era la misma. Al introducir los bloques estos elementos han de transmitirse de un bloque a otro, lo cual puede hacerse con distintas estrategias. Algo parecido ocurre con el ámbito, y la forma en que se manejan los ámbitos en los lenguajes de programación nos servirá de guía para estudiar esas estrategias.

Virtualmente todos los lenguajes cuentan con alguna noción de contexto. El contexto en el que se utiliza una frase influye en su significado. En un lenguaje de programación, los contextos son los encargados de atribuir significado a los identificadores. En semántica denotacional, el contexto de una frase se modela por un mecanismo llamado *ámbito (environment)*. Este concepto no fue necesario en el [4] anterior ya que el lenguaje allí tratado tenía exactamente un ámbito. Este ámbito iba unido al almacén, dando lugar a una aplicación de los identificadores en los valores almacenables. Este modelo tan simple se dividirá en dos componentes, el ámbito y el almacén.

Los ámbitos se utilizan como argumentos en las funciones de valuación. El significado de una sentencia, se determinará, en principio, por la función:

$\mathcal{S}$: **Sentencias** $\rightarrow$ **Amb** $\rightarrow$ **Ind_Fuz** $\rightarrow$ **T_op** $\rightarrow$ **Alm** $\rightarrow \mathcal{P}$(**Alm**)

de forma que para un determinado índice y una **T_op**, el significado de una sentencia es una función $\mathbf{Alm} \rightarrow \mathcal{P}(\mathbf{Alm})$ que es determinada una vez que el ámbito establece el contexto para la sentencia. Por tanto el ámbito pertenecerá al dominio

$$\mathbf{Ambito} = \mathbf{Identificadores} \rightarrow \mathbf{Valores\ Denotables}$$

donde **Valores Denotables** es el dominio de todos los valores que un identificador pueden representar. Para un programa (de un lenguaje determinista) existirá un único almacén y varios ámbitos, los que sean necesarios para establecer los contextos de los diversos bloques, tales como funciones y procedimientos.
Al invocar a una abstracción (función o procedimiento) hay que elegir el contexto en el que dar significados a los identificadores, existen dos posibilidades:

- Elegir el contexto activo en el momento de la invocación
- Elegir el contexto activo en el momento de la declaración de la abstracción

Considerando que **Ind_Fuz** y **T_op** como algo global tendremos las siguientes opciones:
Para **Ind_Fuz**

1. Que las sentencias que componen el cuerpo de la abstracción toman como índice, el índice existente en el momento de la invocación (podríamos decir que tiene alcance dinámico).
2. Que las sentencias que componen el cuerpo de la abstracción toman como índice, el índice que se define en el momento de la declaración de la abstracción (podríamos decir que tiene alcance estático).
3. Incluso existe una tercera vía: que el índice para evaluar las sentencias del cuerpo de la abstracción, sea el resultado de operar, por medio de una T-norma o S-conorma, el índice que existe en el momento de la invocación con el índice que se define en el momento de la declaración de la abstracción.

   Usando las opciones 1 y 3, con los resultados de las abstracciones que se invocan en una ramificación se podrían construir inferencias borrosas. La opción 2 rompe esta posibilidad.

Para **T_op** ocurre algo parecido:

1. Que la **T_op** sea la utilizada en el momento de la invocación de la abstracción.
2. Que la **T_op** sea la definida en el momento de la declaración de la abstracción.

3 Tipos

Vamos a introducir una representación compacta para los conjuntos borrosos, para ello utilizaremos los números trapezoidales, y como deseamos tratarlos directamente, es decir queremos nombrarlos, guardarlos y que puedan ser el resultado de una operación o de la llamada a una función, deberán formar parte tanto de los valores almacenables, como de los denotables y los expresables.

Además queremos dar la posibilidad de crear nuevos tipos, sobre todo para tratar con lo que Zadeh llama *valores lingüísticos* y *variables lingüísticas*[6]. Con ellos se da significado a las frases coloquiales como *poco adecuado*, *adecuado*, *muy adecuado* etc., referidas a algunas característica de un determinado objeto. El objeto, en su forma más simple, vendrá definido a partir de unas características observables en determinadas escalas. Cada una de estas escalas podemos dividirla, de forma borrosa, en distintos tramos que etiquetaremos con un valor lingüístico. El universo del discurso para el objeto será el producto cartesiano de las características. Los subconjuntos borrosos de dicho producto, construidos a partir de operadores lógicos y valores lingüísticos de las características formaran la base a partir de la cual podremos establecer valores lingüísticos para el objeto, ver [1],[2]. Pares de identificadores nos servirán para establecer *antínomos*

Para la declaración de estos nuevos tipo haremos uso del *principio de cualificación* de Tennet[5]. Según dicho principio todo dominio sintáctico puede tener un bloque para admitir declaraciones locales. En particular esto lo aplicaremos a la ampliación de los "registros". Ya que el cuerpo de un registro es una declaración vamos a permitir que ella aparezcan también funciones, lo cual es semánticamente correcto al ser cada registro una especie de ámbito en el que cada identificador está ligado a un valor denotable, por tanto este puede una función. Con lo cual obtendremos una especie de *clase*.

4 Sintaxis abstracta

P ::= K.

K ::= D **begin** S **end**

D ::= **const** D_c^*
| **var** D_v^*
| **type** D_t^*
| **function** I (Π^*) T ; K
| **procedure** I (Π^*) ; K

D_c ::= I = E

D_v ::= I T

D_t ::= I = **tipo_li** Π^* **va_li** $(V_l)^*$ (I I)* **end**

Π ::= I T

V_l ::= I (Π^*) T ; K
| **as** I

S ::= I **:=** E
| I(E*)
| **case** G **esac**
| **do** G **od**
| K
| **skip**
| **return**
| I_1 **<-** I_2
| S ; S

G ::= E -> S | G □ G

E ::= I | N | B | E Ω E | ϒ E | I(E*) | I.E

5 Álgebras semánticas

Fundamentalmente vamos a describir la semántica denotacional dinámica. En dicha semántica es irrelevante el asignar un símbolo para representar los errores, pues suponemos que los programas que los contengan ha sido rechazados por ilegales. Por ello reservamos $\perp$ para representar la *no terminación* y no se introducirá ningún símbolo para representar errores en todos los dominios e incluso no especificaremos el tratamiento de dichos errores.

I Literales

(Literal)
L ::= B | N | R | G | C | CC
Booleanos
Bool ::= true | false
(Numeral)
Num ::= *sin especificar*
Numeral Real
Real ::= *sin especificar*
Grado
Grd ::= *sin especificar*
(Carácter)
C ::= *sin especificar*
(Cadena-Caracteres)
CC ::= *sin especificar*

II Identificadores

III Booleanos Borrosos
Dominio **BoolB = Bool** $\times$ **Grd**

IV Números Borrosos
Dominio **NumB = Num** $\times$ **Grd**

V Reales Borrosos
Dominio **RealB = Real** $\times$ **Grd**

VI Números Trapezoidales
Dominio **NumT = Real** $\times$ **Real**$\times$ **Real**$\times$ **Real**$\times$ **Grd**
Operaciones

- sumt, dift, multt, divt : **NumT** $\otimes$ **NumT** $\circ\!\!\rightarrow$ **NumT**

VII Posiciones de memoria
Dominio **Loc** con las operaciones usuales: primera_locn, siguiente_locn, igual_locn, menorque_locn

VIII Valores expresables
Deseamos que **NumT** sea de "primera categoría", es decir que pueda pasarse como parámetro a una función o ser devuelto por ella, etc.

Dominio **VE = BoolB** $\oplus$ **NumB** $\oplus$ **RealB** $\oplus$ **NumT**.

IX Valores Denotables
El conjunto de valores que pueden ser "denotados" por identificadores es:

Dominio **VD = Const** $\oplus$ **Loc** $\oplus$ **Abst** $\oplus$ **VaL** $\oplus$ **TiL**
donde
Const = BoolB $\oplus$ **NumB** $\oplus$ **RealB** $\oplus$ **NumT**
y
Abst = Func $\oplus$ **Proc**

X Funciones

Func = Param $\circ\!\!\rightarrow$ **Ind** $\circ\!\!\rightarrow$ **Alm** $\circ\!\!\rightarrow$ (**VE** $\times$ (**Alm**$_\perp$ $\oplus$ δ))$^\natural$

XI Procedimientos

Proc = Param $\circ\!\!\rightarrow$ **Ind** $\circ\!\!\rightarrow$ **Alm** $\circ\!\!\rightarrow$ (**Alm**$_\perp$ $\oplus$ δ)$^\natural$
donde **Param = VE**

XII Variables lingüísticas

VaL = Ide $\rightarrow$ (**Loc** $\oplus$ **Func** $\oplus$ **VaL**)

XIII Tipos lingüísticos

TiL = Amb $\circ\!\!\rightarrow$ **Alm** $\circ\!\!\rightarrow$ (**VD** $\times$ **Alm**)

XIV Ámbitos

Al dotar a nuestro lenguaje de estructura de bloques, necesitamos los ámbitos, para representar asociaciones entre los identificadores y los valores denotados. El elemento $\top \in \mathbf{O}$ se utiliza para indicar la ausencia del valor denotado

Dominio $\mathbf{Amb} = \mathbf{Ide} \rightarrow (\mathbf{VD} \oplus \mathbf{O})$

Operaciones Serán las usuales: vac_amb, acc_amb, cre_amb, sol_amb, mod_amb, une_amb.

XV Valores almacenables

Para representar el conjunto de valores que pueden ser almacenado en una única localización, para ello se utiliza el

Dominio $\mathbf{VA} = \mathbf{BoolB} \oplus \mathbf{NumB} \oplus \mathbf{RealB} \oplus \mathbf{NumT}$

XVI Almacén: memoria basada en una pila

Los "almacenes" se utilizan para representar asociaciones entre las variables y sus valores. Las variables son representadas por "localizaciones" en los almacenes y la única propiedad relevante que poseen es la de ser distinguibles una de otra. Por tanto el identificador de una variables está ligado a una localización, que a su vez da acceso al valor almacenado en la variable. Ello hace que a veces se introduzcan los dominios **LV** de todas las variables y **RV** el dominio de valores asignables. Generalmente a la hora de administrar los almacenes solo se necesita saber si una determinada localización está reservada o no. Lo cual hace que por ejemplo la función **asi_loc** se deje sin especificar y todo el modelo se simplifique.

Dominio $\mathbf{Alm} = \mathbf{Loc} \rightarrow (\mathbf{VA} \oplus \mathbf{O}) \times \mathbf{Loc}$

Operaciones Serán las usuales: vac_alm, acc_alm, mod_alm, mar_loc, asi_loc, des_loc

Programa Ejemplo

```
\       Funciones parecido y alrededor.
\         Para estrategias 1,2,3 usar _y
\         para estrategias 4,5,6 usar _y1
const _y = {0.,0.,0.,0.};
const _y1 = {1.,1.,1.,1.};
function parecido(ENTERO x1): BORROSO
const
    a1= 5., b1=10.;

var
    BORROSO z;
    REAL co;
begin
    if NEUTRO = 1.0 then z:=_y else z:=_y1 end if;
    co := EXTR(x1);
    z{1}:=co-a1-b1;
    z{2}:=co-a1;
    z{3}:=co+a1;
    z{4}:=co+a1+b1;
    parecido <- z
end;

function alrededor(ENTERO x1): BORROSO
const
    b1= 4., a1=5.;

var
    BORROSO z;
    REAL co;
begin
    if NEUTRO = 1.0 then z:=_y else z:=_y1 end if;
    co := EXTR(x1);
    z{1}:=co-a1-b1;
    z{2}:=co-a1;
    z{3}:=co+a1;
    z{4}:=co+a1+b1;
    alrededor <- z
end;
end
```

Referencias

[1] J.M. Adamo. L.p.l. a fuzzy programming language: 1. syntactic aspects. *Fuzzy Set and Systems*, 3:151–179, 1980.

[2] J.M. Adamo. L.p.l. a fuzzy programming language: 2. semantic aspects. *Fuzzy Set and Systems*, 3:261–289, 1980.

[3] Daniel Sanchez Alvarez. El lambda cálculo_b. *En Actas del VII Congreso Español sobre Tecnología y Lógica Fuzzy. pag.311-316*, 1997.

[4] Daniel Sanchez Alvarez y Antonio F. Gómez Skarmeta. Semánticas de lenguajes con datos borrosos. *En Actas del VIII Congreso Español sobre Tecnología y Lógica Fuzzy. pag.271-278*, 1998.

[5] R.D. Tennent. *Principles of Programming Languages.* Prentice-Hall, Englewood Cliffs,N.J., 1981.

[6] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning-I. *Information Sciences*, 8:199–249, 1975.