ANOTACIONES DE MERLIN, COMPORTAMIENTO DE UNIVERSOS Y SEMANTICA ALGEBRAICA

Nivela, Ma. P. y Orejas, F.

Facultad de Informática de Barcelona.

Resumen. En este trabajo se presentan las ideas básicas seguidas para el diseño de un lenguaje de anotaciones para el lenguaje de programación Merlín. En concreto, las anotaciones se prevén en forma de especificaciones ecuacionales ligadas a los universos (construcción modular básica de Merlín), tanto a su componente de interfaz como a su componente de implementación. De cara a definir la semántica del lenguaje de anotaciones se ha optado por una semántica algebraica de comportamiento, en la que ciertas componentes son interpretadas inicialmente.

Los tipos abstractos de datos son actualmente una de las bases más sólidas para el diseño sistemático, modular y fiable de programas. El enfoque algebraico (especificaciones ecuacionales) se ha impuesto como la forma más apropiada de describir con rigor y coherencia un tipo de datos, al mismo tiempo que se posibilita la demostración de propiedades y la verificación de implementaciones.

Para dar un sentido preciso y sin ambigüedades a una especificación es necesario proporcionar una definición formal de su semántica. Existen diversos enfoques para dar semántica a una especificación: inicial (ADJ78, ADJ81, EKMP82, EH82), final (WA79, KA83), laxa (BPW 81) y comportamiento (REI81, GM82).

En este momento, el Depto. de Programación de Computadores de la Fac. de Informática de Barcelona se propone la ampliación del lenguaje de programación Merlín (BO82, BOP82) incorporándole aspectos de especificación, en la misma linea que el lenguaje de anotaciones Anna (KL80) extiende a Ada. En nuestro caso las anotaciones se restringirían a la especificación ecuacional de los universos, tanto de su interfaz como de su implementación.

El objetivo de las anotaciones no es tanto el de posibilitar la verificación (automática o semi-automática) de programas Merlín, como el de posibilitar la validación de su diseño,cuando éste se encuentra todavia en fase de especificación, utilizando técnicas de reescritura.

Se ha escogido la semántica de comportamiento como la más adecuada para la definición formal del lenguaje de anotaciones, ya que es la que caracteriza mejor la idea de que un módulo (universo) es una caja negra de la cual sólo nos interesa su comportamiento observable. Sin embargo, para establecer el criterio de observabilidad, es necesario interpretar alguna parte de la especificación de forma inicial. Esto corresponde a un caso particular del concepto de "initial restriction" (HKR80), "data constraint" (BG80) y "free constraint" (EWT82).

La organización de este trabajo es la siguiente: en la primera sección se presenta la estructura básica del lenguaje Merlín y de lo que será su lenguaje de anotaciones. En la segunda sección se presentan las construcciones semánticas básicas para establecer la definición formal del lenguaje de anotaciones. Finalmente, en la tercera sección se extraen algunas conclusiones y se comentan algunos problemas todavia no resueltos.

1. UNIVERSOS DE MERLÍN

Merlín es un lenguaje de programación imperativo, fuertemente compacto en torno a una idea básica, la construcción de programas utilizando el concepto de tipo abstracto de datos como forma básica de abstracción. Como consecuencia, su construcción fundamental es el universo. En un universo pueden ser definidos tipos (con sus operaciones) o enriquecimientos a tipos ya existentes, (declarados con la

do de nuestra especificación, entre ellos, modelos triviales.

Si eligiéramos semántica final, en principio, tendríamos el significado correcto, ya que habríamos especificado el tipo conjunto. Sin embargo, consideramos la semántica final demasiado restrictiva: otros modelos cuyo comportamiento observable sea igual al de los conjuntos tambien han de ser modelos válidos: ésto nos lo da la semántica de comportamiento.

Ahora bien, para definir un comportamiento es necesario considerar una parte de la especificación como observable. Esto lo hemos hecho interpretando inicialmente una subespecificación de la especificación dada. En el ejemplo anterior, los booleanos serían interpretados inicialmente. En concreto:

Definición 1

Una especificación de comportamiento BSP es un par de especificaciones (SP1,SP2), SPi = (Σi,Ei), con SP1 SP2. SP1 se denomina parte observable de BSP.

Definición 2

Sea $\Sigma 1 \subseteq \Sigma 2$, y sean A1 y A2 dos $\Sigma 2$-álgebras, se dice que A1 es observacionalmente equivalente a A2 con respecto de $\Sigma 1$, A1 $\equiv_{\Sigma 1}$ A2, sii:

1) $A|_{\Sigma 1} - A|_{\Sigma 2}$

2) Existen dos valuaciones v1 y v2, v1: X ---> A1 y v2: X ---> A2 tal que para todo par de $\Sigma 2(X)$-términos t1 y t2, con sus géneros y los géneros de X en S1, se verifica que:

$A \models v1(t1)=v1(t2)$ sii $B \models v2(t1)=v2(t2)$

Esta definición de equivalencia observacional es una variante de la introducida por Sannella y Wirsing (SW 83) en la que se consideran ecuaciones de términos de una signatura derivada (como términos a observar) y en la que, en cierta manera, aparece una idea de restricción.

Denominaremos Comp($\Sigma 2$,$\Sigma 1$) a las clases de equivalencia de $\Sigma 2$-álgebras (con respecto de $\equiv_{\Sigma 1}$).

Se puede demostrar que todo comportamiento contiene un álgebra final. En consecuencia, de ahora en adelante, identificaremos los comportamientos con sus álgebras finales. Mas aún, se puede definir un funtor Beh que asocia a cada $\Sigma 2$-álgebra A el álgebra final de su comportamiento. En concreto, Beh(A) sería el cociente de A por una congruencia de Nerode.

Definición 3

Un comportamiento B de Comp($\Sigma 2$,$\Sigma 1$) satisface una especificación de comportamiento BSP = (SP1,SP2) sii existe un álgebra A en B, tal que A satisface SP2.

Proposición 4

Si B satisface BSP entonces el álgebra final de B satisface SP2.

En adelante, denominaremos COMP(BSP) a la subcategoria de Alg_{SP2} formada por las álgebras (junto con sus homomorfismos) que son finales en algun comportamiento que satisface BSP.

El siguiente teorema nos permite dar la semántica requerida a las especificaciones de comportamiento no parametrizadas:

Teorema 5

Sea BSP = (SP1,SP2) una especificación de comportamiento, entonces existe un comportamiento inicial en COMP(BSP).

Por falta de espacio, no incluimos la demostración de este u otros resultados. Para mas detalles, vease la versión larga de este trabajo. De todas maneras, la idea básica de la demostración consiste en probar que dicho comportamiento inicial es

$$Beh(T_{SP2}).$$

Definición 6

Una especificación de comportamiento parametrizada BPSP es un par (BSP1, BSP2) de especificaciones de comportamiento. BSP1 es la especificación del parámetro formal y BSP2 la especificación del cuerpo de la parametrización. Evidentemente, BSP1 $\subseteq$ BSP2.

Un resultado similar al anterior nos permite dar la semántica adecuada a las especificaciones parametrizadas. En concreto:

Teorema 7

Dadas dos especificaciones de comportamiento BSP1 y BSP2, con BSP1 BSP2, existe un funtor libre F de COMP(BSP1) a COMP(BSP2), adjunto por la izquierda al funtor olvidadizo U de COMP(BSP2) en COMP(BSP1).

Este resultado puede ser enunciado de forma mas general pidiendo, solamente, la existencia de un morfismo de especificaciones (de comportamiento) entre BSP1 y BSP2, en lugar de la inclusión. La idea de la demostración consiste en probar que:

cláusula usa). Además, un universo puede ser genérico o parametrizado; sus parámetros son tipos (eventualmente, con operaciones) declarados con la cláusula requiere. Un universo genérico puede ser instanciado utilizando la cláusula donde.

Con respecto a la cuestión de la visibilidad, un universo consta de dos partes: una visible (mal llamada) de especificación, que sirve de interfaz con el exterior, y una zona oculta de implementación, en la que se declara como se representan los tipos definidos en el universo y se implementan sus operaciones.

Por ejemplo, el universo genérico conjunto de valores, podría ser:

```
Universo conj_de_valores
   usa bool
   requiere tipo valor
            funcion eq(valor,valor)
                    devuelve bool
   *** 1 ***
   tipos conjunto
   ops
     accion vacio(conjunto);
     accion poner(conjunto,valor);
     funcion pert(conjunto,valor)
             devuelve bool;
     *** 2 ***
   fops
   impl
     rep
       tipo conjunto es  ....
     frep
     accion vacio(C) es
       ...
     funcion  pert(C,V) es
       ...
     *** 3 ***
funiverso
```

Como ya se ha dicho, se pretende extender el lenguaje Merlín con anotaciones. Estas anotaciones serían, en principio, conjuntos de ecuaciones que podrían ser colocadas en los puntos 1, 2 y 3. En 1, las ecuaciones definirían las propiedades que se requieren a los posibles parámetros actuales. Por ejemplo, que eq sea una operación de igualdad y que, por tanto, cumpla las propiedades reflexiva simétrica y transitiva. En 2, se especificaría el universo, ésto es, los tipos definidos en él. Finalmente, con las ecuaciones de 3 se especificaría la implementación del universo.

En concreto, las ecuaciones que anotarían el universo conjunto de valores podrían ser:

en *** 1 ***

1.1 eq(X,X) = true

1.2 eq(X,Y) = eq(Y,X)

1.3 (eq(X,Y) & eq(Y,Z)⇒eq(X,Z))=true

en *** 2 ***

2.1 poner(poner(C,X),Y) =
    poner(poner(C,Y),X)

2.2 pert(vacio,X) = false

2.3 pert(poner(C,X),Y) =
    pert(C,Y) or eq(X,Y)

Las ecuaciones a colocar en 3 dependerían del tipo utilizado para la representación de los conjuntos. Por ejemplo, si se toman listas, estas ecuaciones pueden verse en (SW83).

Las anotaciones, además de como comentarios y de su posible papel en la verificación de programas Merlín, en nuestro contexto se utilizarán para la validación de los diseños en etapas tempranas de los mismos. En efecto, la utilización de herramientas basadas en la reescritura de términos posibilita, de una parte, la ejecución de las especificaciones lo que permite, en ese momento, contar con prototipos del programa diseñado, y, por otra parte, verificar (en forma automática o semiautomática) la corrección del diseño, contrastando las especificaciones de cada universo con las de su implementación.

2. SEMÁNTICA DE LAS ANOTACIONES

Como ya se indicó en la introducción, pueden definirse varias semánticas diferentes para una misma especificación algebraica: inicial, final, laxa o las varias posibles semánticas de comportamiento. Hemos elegido un tipo de semántica de comportamiento como la más adecuada para nuestro lenguaje de anotaciones. Veamos porqué.

Si en el ejemplo anterior hubiéramos tomado semántica inicial,no habríamos especificado los conjuntos de valores, sino los multiconjuntos. Para especificar los conjuntos habríamos tenido que añadir la ecuación:

meter(meter(C,X),X) = meter(C,X)

lo que,en principio, consideramos innecesario, ya que esta ecuación describe una propiedad del tipo conjunto que no es directamente observable desde el exterior.

Si eligiéramos semántica laxa, tendríamos demasiados modelos como significa-

$$F = Beh \cdot G$$

donde G es el funtor libre de Alg_{SP12} en Alg_{SP22}.

Esto es, en nuestro lenguaje de anotaciones, definiremos como semántica de una especificación no parametrizada al comportamiento inicial en la clase de comportamientos que satisfacen la especificación; y como semántica de una especificación parametrizada al funtor libre asociado a la inclusión del parámetro formal en el cuerpo de la parametrización. La parte observable de una especificación se considerará formada por las especificaciones de los tipos standard de Merlín (enteros, reales, booleanos y caracteres).

3. CONCLUSIONES

Se han dado las ideas básicas para definir un lenguaje de anotaciones que extienda Merlín, y para definir su semántica algebraica de comportamiento.

De cara a completar el trabajo, quedaría precisar la relación que ha de haber entre las (semánticas de las) anotaciones de especificación y de implementación de un universo.

También quedaría comprobar que, con este nuevo tipo de semántica, se cumplen los principios de composición horizontal y vertical del sistema CAT (GB80).

AGRADECIMIENTOS

Este trabajo está subvencionado por la Comisión Asesora de Investigación Científica y Técnica (ref. 2704-83).

4. REFERENCIAS

ADJ78 Thatcher, J.W.; Wagner, E.G. Wright, J.B.:'Data type specification: parameterization and the power of specification techniques', TOPLAS 4,4 (1982), pp. 711-732.

ADJ81 Ehrig H. et al.: 'Parameter passing in algebraic specification languages', Proc. Workshop on Program Specification, LNCS 134 (1982), pp. 322-369.

BO82 Botella, P.; Orejas, F.: 'Merlí: report preliminar', FIB - 1982.

BOP82 Botella, P.; Orejas, F.; Pueyo X.:'Hacia un entorno de programación basado en tipos de datos: la notación Merlín'. Actas V Congreso AEIA, pp. 293-299, 1982.

BG80 Burstall, R.; Goguen J.A.:'The semantics of Clear, a specification language', LNCS 86, 1980, pp. 294-332

KL80 Krieg-Brückner, B.;Luckham, D.C 'Anna: towards a language for annotating Ada programs', Proc. Ada Conf. 1980.

BPW81 Broy, M; Pair, C.; Wirsing, M: Asystematic study of models of abstract data types', CRI Nancy Rep. 81-R-042, 1981.

EH82 Ehrich, H.-D.:'On the theory of specification, implementation and parameterization of abstract data types' JACM 29,1 (1982), 206-227.

EKMP82 Ehrig, H. et al: 'Algebraic implementation of abstract data types' TCS 20 (1982), pp. 209-263.

EWT82 Ehrig, H.; Wagner, E.G; Thatcher, J.W.:' Algebraic constraints for specifications and canonical form results', TU Berlin rep. 82-09.

GM82 Goguen, J.A., Meseguer, J.: 'Universal realization, persistent interconnection and implementation of abstract modules', proc 9º ICALP LNCS 140, pp. 265-281, 1982.

GB80 Goguen, J.A.; Burstall, R.M.: 'CAT a system for the structured elaboration of correct programs from structured specifications', SRI Int. Rep. CSL-118, 1980.

HKR80 Hupbach, V.L.; Kaphengst, M.; Reichel, H.: Initial algebraic specification of data types, parameterized data types and algorithms', VEB Robotron ZFT Tech. rep. Dresden 1980.

KA83 Kamin, S.: 'Final data types and their specification', TOPLAS 5,1 pp. 77-123, 1983.

REI81 Reichel, H.: 'Behavioural equivalence: a unifying concept for initial and final specifications', 3rd. Hungarian Conf. on Comp. Sc., 1981.

SW83 Sannella, D.; Wirsing, M.: 'A kernel language for algebraic specification and implementation', Springer LNCS 158, 1983.

WA79 Wand, M.: ' Final algebra semantics and data type extensions', JCSS 19,1 pp. 27-44.

SUMMARY

The basic ideas for the design of an annotation language for the programming language Merlin are presented. A new semantics based on behaviour is introduced to formally define annotations.