ESPECIFICACION FORMAL DE UN SISTEMA BASADO EN REGLAS*

J. Agustí-Cullel y G. Valiente

Abstract

As expert systems become more complex, and development teams larger, problems similar in nature to those of software development for computer science arise. Software engineering gave an answer to such problems in computer science, by applying formal methods to the development process and by offering better support tools. In a similiar vein, we propose the application of formal methods and techniques to the specification, design, implementation, validation, and maintenance of expert systems. We show the formal specification of a simplified expert system and its implementation by stepwise refinement in an algebraic specification language, demonstrating the usefulness of such formal methods in relation to expert system development.

1. Introducción

El desarrollo de un sistema experto es actualmente un proceso evolucionario, por el cual se construye una versión preliminar del sistema y se refina progresivamente, hasta alcanzar un estado final aceptable. Durante este proceso se toman decisiones de diseño ad-hoc, resultando en sistemas sin una clara separación entre el conocimiento de dominio y el de implementación, difíciles de comprender, mantener y validar.

Un problema similar fue atacado a mediados de la década del 70 para el desarrollo de sistemas de software, cuando surgió la disciplina de la ingeniería de software como respuesta a la denominada crisis del software. Esta disciplina se caracteriza por el amplio uso de técnicas y herramientas formales en todas las fases del ciclo de desarrollo de software. Desde entonces, las técnicas y los lenguajes de especificación algebraica se han aplicado satisfactoriamente para la especificación de todo tipo de sistemas, desde tipos abstractos de datos hasta sistemas de software muy complejos.

En este artículo analizamos la adaptabilidad de los métodos formales a la especificación de sistemas de inteligencia artificial en general y de sistemas expertos en particular. Exploramos la utilidad de las especificaciones algebraicas en relación al desarrollo de sistemas expertos, mediante un caso de estudio.

* Trabajo realizado bajo Contrato 880J382 de CICYT-CSIC.

AGORA 9 (1990): 23-30

A la hora de seleccionar un caso de estudio adecuado, nos enfrentamos a un compromiso entre la complejidad y la comprensibilidad del problema a atacar. Un problema pequeño es fácil de entender, y por lo tanto puede usarse para enfatizar cuestiones metodológicas, pero resulta difícil convencer a gente experimentada sobre la base de un problema irrealista o de juguete. Un problema grande, por otro lado, requiere un gran esfuerzo para explicar el problema en sí mismo, para analizar cuestiones técnicas, y resulta fácil perder de vista los aspectos metodológicos.

Hemos escogido como caso de estudio un sistema basado en reglas [9] [5] [4] simplificado, que puede verse como una solución de compromiso entre problemas de juguete y problemas realistas. Es realista, pues es un subconjunto de varios sistemas expertos existentes, pero no es demasiado grande, permitiendo mostrar principios metodológicos sin perderse en detalles técnicos específicos al problema.

Nuestro caso de estudio consiste en el desarrollo de un sistema basado en reglas simples. Para dar una descripción informal del sistema, podemos considerar a un sistema basado en reglas como compuesto por una base de conocimientos y un motor de inferencias. La base de conocimientos contiene hechos y reglas. Un hecho es una proposición cierta en el dominio de aplicación del sistema. Una regla expresa un condicional, con una parte antecedente y una parte consecuente.

La interpretación de una regla es que si se puede satisfacer el antecedente, entonces también se puede satisfacer el consecuente. Cuando el consecuente define una acción, el efecto de satisfacer el antecedente es el de planificar la ejecución de la regla. Cuando el consecuente define una conclusión, el efecto de satisfacer el antecedente es el de inferir la conclusión.

Durante una sesión con el sistema basado en reglas, una memoria de trabajo registra el estado del sistema durante la solución de un problema. La memoria de trabajo contiene hechos deducidos y objetivos pendientes. Una sesión de solución de problemas se puede ver como el proceso de completar una base de conocimientos incompleta en una memoria de trabajo, que inicialmente puede estar vacía, dirigido por el conocimiento contenido en la base de conocimientos y por la interacción con el mundo esterior, i.e., hechos ingresados desde sensores, hechos y/o objetivos ingresados por el usuario como respuestas a las preguntas del sistema, etc.

Las características funcionales de un sistema basado en reglas se verán en más detalle durante el proceso de especificación formal.

2. Especificaciones formales

La necesidad de la especificación formal de un sistema de software puede verse por analogía con la construcción de una casa. Cuando alguien desea construir una casa, le conviene contratar un arquitecto para asegurarse que lo que obtiene construido es lo que realmente desea. El arquitecto prepara una especificación detallada de la casa, que se puede discutir y corregir antes de comenzar su construcción. Del mismo modo, antes de invertir esfuerzo alguno en desarrollar un programa de software también conviene elaborar una especificación, para asegurarse de que lo que se produce sea exactamente lo requerido. Las especificaciones formales tienen la ventaja de ser no

ambiguas, y además permiten conocer las propiedades de una especificación.

Una especificación formal es, en principio, una especificación expresada en un lenguaje formal. Existen numerosos enfoques y lenguajes para la especificación formal de software [6]. En este artículo seguiremos la metodología de especificación algebraica, tal como la representan los lenguajes Extended ML y Standard ML. Extended ML [7] es un lenguaje de amplio espectro, y Standard ML [3] es un lenguaje de programación funcional que incorpora un sistema de tipos seguro y extensible que incluye tipos polimórficos, un sistema de excepciones potente, alguns constructos imperativos y facilidades poderosas para la modularización de programas.

Una especificación formal de Standard ML consiste de un conjunto de tipos (sorts) y un conjunto de funciones (operaciones) sobre esos tipos. Los tipos se pueden definir en términos de tipos más primitivos usando constructores primitivos o bien constructores definidos por el usuario, y el lenguaje provee de un conjunto de tipos primitivos. Las funciones se definen mediante ecuaciones de una cierta forma restringida, y esa restricción sintáctica es lo que permite que los programas sean ejecutables. El lenguaje también permite definir funciones de orden superior, que toman funciones como argumentos y/o producen funciones como resultado.

El lenguaje Standard ML permite la definición por separado de la interface de un módulo de programa (signatura) y de su implementación (estructura). Cada estructura tiene una signatura asociada, que declara los nombres de los tipos y funciones definidas en la estructura. Las estructuras se pueden construir sobre otras estructuras existentes, de modo que cada estructura es en realidad una jerarquía de estructuras, y esto se refleja también en su signatura. Para permitir la interacción entre diferentes módulos de un programa, es posible declarar que ciertas subestructuras en una estructura son idénticas (compartidas). El lenguaje también permite definir estructuras parametrizadas (functores). La aplicación de un functor a una o varias estructuras produce otra estructura. Un functor tiene una signatura de entrada, que describe las estructuras que puede recibir como argumento, y una signatura de salida que describe el resultado de una aplicación del functor.

Una especificación formal en Extended ML se puede obtener extendiendo una especificación en Standard ML mediante el agregado de axiomas a las signaturas y a las definiciones de estructuras y de functores. Los axiomas en una signatura restringen el comportamiento permitido a los componentes de las estructuras que concuerdan con la signatura. Los axiomas en una estructura o en un functor se emplean para definir datos y funciones a un alto nivel de abstracción, que no necesariamente es ejecutable. Algunas especificaciones en Extended ML son ejecutables, ya que las definiciones de funciones en Standard ML son simplemente axiomas de una cierta forma restringida. En estos términos, el objetivo del desarrollo de programas es refinar especificaciones no ejecutables hasta que solamente contengan axiomas ejecutables.

3. Especificación Algebraica de un Sistema Basado en Reglas

Para desarrollar una especificación algebraica de nuestro caso de estudio, hemos de identificar en primer lugar los tipos de objeto básicos y las principales funciones

que operan sobre ellos. Los principales tipos son los siguientes:

he: Tipo de hechos.

re: Tipo de reglas de producción. Una regla de producción suele consistir de una lista de hechos antecedente y de un único hecho consecuente.

ob: Tipo de objetivos a alcanzar. Consideraremos como objetivos a los hechos a ser deducidos por el sistema.

bc: Tipo de bases de conocimiento. Una base de conocimientos representa el conocimiento específico al dominio de una aplicación particular del sistema.

mt: Tipo de estados de la memoria de trabajo. Un estado dado de la memoria de trabajo representa un cierto punto hacia la solución del problema planteado al sistema.

Las funciones principales serán definidas en forma descendente. La función de nivel superior es *motor*, que toma una base de conocimientos y un estado de la memoria de trabajo como argumentos y transforma la última mediante la invocación repetitiva de un ciclo reconocimiento-acción [1]. Este ciclo consiste de tres fases: (a) buscar reglas aplicables al estado actual de la memoria de trabajo, (b) resolver conflictos ordenando las reglas aplicables según algún criterio, y (c) ejecutar las reglas en ese orden. El ciclo se repite hasta alcanzar una cierta condición de parada, tal como la ausencia de objetivos pendientes. Por lo tanto, la signatura de la función *motor* es la siguiente:

$$\text{motor} : \text{bc} \times \text{mt} \rightarrow \text{mt}$$

La cuestión central a la especificación de esta función es el significado de que una regla sea aplicable a un cierto estado de la memoria de trabajo, y el significado de la ejecución de una regla. Los significados más usuales originan los motores de inferencia con encadenamiento hacia adelante y con encadenamiento hacia atrás.

En un motor hacia adelante, una regla es aplicable a un estado dado de la memoria de trabajo si su antecedente está incluido en la lista de hechos en ese estado de la memoria de trabajo, y su ejecución es de antecedente a consequente: siempre que una regla sea aplicable, el resultado de su ejecución es inferir el consecuente.

En un motor hacia atrás, en cambio, una regla es aplicable a un estado dado de la memoria de trabajo si su consecuente pertenece a la lista de objetivos pendientes en ese estado de la memoria de trabajo, y su ejecución es de consequente a antecedente: siempre que una regla sea aplicable, el resultado de su ejecución es planificar los hechos antecedentes como subobjetivos a lograr. Por lo tanto, cabe diferenciar entre estos dos tipos de motores de inferencia, si bien sus signaturas son idénticas a la anterior:

$$\text{adelante} : \text{bc} \times \text{mt} \rightarrow \text{mt}$$

atrás : bc × mt → mt

Para especificarlos, se necesita al menos una función para cada fase del ciclo reconocimiento-acción. En el motor hacia adelante, la función *adel-aplicables* toma como argumento la lista de reglas en la base de conocimiento y la lista de hechos en el estado actual de la memoria de trabajo, y produce como resultado la lista de reglas aplicables a ese estado de la memoria de trabajo:

adel-aplicables : re list × he list → re list

La función *adel-conflicto* toma la lista de reglas aplicables y las ordena según un cierto criterio, tal como tamaño, complejidad, subsunción, etc.:

adel-conflicto : re list → re list

La función *adel-ejecución* toma la lista de hechos en el estado actual de la memoria de trabajo y una regla, y ejecuta la regla, produciendo como resultado una lista de hechos inferidos, que serán agregados a la lista de hechos del estado actual de la memoria de trabajo. Solamente se agregan nuevos hechos, que no existan previamente en la lista de hechos del estado actual de la memoria de trabajo:

adel-ejecución : he list × re → he list

En el motor hacia atrás, la función *atr-aplicables* toma la lista de reglas de la base de conocimientos y un objetivo del estado actual de la memoria de trabajo, y produce la lista de reglas aplicables en ese estado de la memoria de trabajo:

atr-aplicables : re list × ob → re list

La función *atr-conflicto* toma la lista de reglas aplicables y las ordena según un cierto criterio, como en el caso del motor hacia adelante:

atr-conflicto : re list → re list

La función *atr-ejecución* toma la lista de objetivos del estado actual de la memoria de trabajo y una regla, y ejecuta la regla, produciendo como resultado una lista de objetivos, que serán agregados a la lista de objetivos pendientes del estado actual de la memoria de trabajo. Solamente se agregan nuevos objetivos, que no existan previamente en la lista de objetivos del estado actual de la memoria de trabajo:

atr-ejecución : ob list × re → ob list

La semántica de las funciones anteriores puede definirse de varias maneras. Una forma de hacerlo es restringiendo su comportamiento mediante el agregado de axiomas. Por ejemplo, los siguientes axiomas, escritos en Extended ML, definen algunas

propiedades de la función *adel-aplicables*.

axiom ∀ RE : re list, HE : he list, R ∈ RE.
satisfactible (HE, antecedente (R)) ⇒ R ∈ adel-aplicables (RE,HE)

axiom ∀ RE : re list, HE : he list.
RE = ∅ ⇒ adel-aplicables (RE,HE) = ∅

axiom ∀ RE1 : re list, RE2 : re list, HE : he list.
RE1 ⊆ RE2 ⇒ adel-aplicables (RE1,HE) ⊆ adel-aplicables (RE2,HE)

Otra forma de definir la semántica de las funciones es mediante una especificación ejecutable escrita en Standard ML. Por ejemplo, la siguiente es una de las posibles especificaciones ejecutables de la función *adel-aplicables*:

```
fun adel-aplicables Reglas Hechos =
    if satisfactible Hechos (antecedente (head Reglas))
    then (head Reglas) cons (adel-aplicables (tail Reglas))
    else adel-aplicables (tail Reglas)
```

Esta función selecciona las reglas aplicables en un estado dado de la memoria de trabajo. El resultado de su aplicación es la lista de las reglas satisfactibles por la lista de hechos del estado actual de la memoria de trabajo.

La definición de esta función en Standard ML es una implementación de su definición en Extended ML, es decir, respeta todos los axiomas de su especificación algebraica.

4. Especificación Ejecutable de un Sistema Basado en Reglas

En la especificación ejecutable final del caso de estudio, hemos aprovechado las facilidades de modularización de programas del lenguaje Standard ML, organizando todas las definiciones previas de tipos y de funciones en una jerarquía de módulos de programa. Las signaturas resultantes son las siguientes:

```
signature Sintaxis =
sig
    type  he
    type  re
    type  ob
    type  bc
    type  mt
    val   antecedente : re → he list
    val   consecuente : re → he
end
```

```
signature forward =
sig
   structure S = Sintaxis
   val    aplicables : re list × he list → list
   val    conflicto : re list → re list
   val    ejecución : mt × re → mt
   val    paso-adelante : bc × mt × re → mt
   val    adelante : bc × mt → mt
end

signature backward =
sig
   structure S = Sintaxis
   val    aplicables : re list × ob → re list
   val    conflicto : re list → re list
   val    ejecución : ob list × re → ob list
   val    paso-atrás : bc × mt × ob → mt
   val    atrás : bc × mt → mt
end
```

5. Conclusión

Hemos mostrado la aplicación de técnicas y lenguajes de especificación albegraica a la especificación formal de un sistema experto simplificado. Las ventajas de este enfoque son similares a las de la especificación formal de sistemas de software en general, es decir, la posibilidad de corregir errores en las primeras etapas del proceso de desarrollo, la comprensibilidad, y el entendimiento del problema durante el proceso de especificación formal.

La experiencia de especificar este sistema experto simplificado se está aplicando a la reconstrucción racional del sistema MILORD [2] [8], y al diseño del lenguaje COLAPSES (Concurrent Language for Parallel Expert System Specification).

Reconocimientos

La idea de aplicar técnicas y lenguajes de especificación algebraica a la especificación formal de sistemas expertos es original de D. Sannella y F. Esteva, a quienes estamos agradecidos.

Referencias

[1] H. Farreny, *Les Sistemes Experts*, Cepadues-Editions, Toulouse, 1985.

[2] L. Godo, R. López de Mántaras, C. Sierra y A. Verdaguer, MILORD: «The

Architecture and the Management of Linguistically Expressed Uncertainty». A aparecer en: *International Journal of Intelligent Systems*.

[3] R. Harper, D. MacQueen y R. Milner, *Standard ML, Report ESC-LFCS-86-2*, University of Edinburgh, Marzo, 1986.

[4] F. Hayes-Roth, «Rule-Based Systems», *Communications of the ACM*, Volúmen 28, Número 9, Septiembre, 1985.

[5] F. Hayes-Roth, D.A. Waterman y D.B. Lenat, *Building Expert Systems*, Addison-Wesley, 1983.

[6] D.T. Sannella, *A Survey of Formal Software Development Methods, Report ECS-LFCS-88-56*, University of Edinburgh, Julio, 1986.

[7] D.T. Sannella y A. Tarlecki, «Extended ML: An Institution-Independent Framework for Formal Program Development», *Proc. Workshop on Category Theory and Computer Programming*, Guildford, Springer-Verlag LNCS 240, pp. 364-389.

[8] C. Sierra, *A Multi-Level Architecture for Expert Systems in Classification*, Tesis Doctoral, Universidad Politécnica de Cataluña, España, Mayo, 1989.

[9] D.A. Waterman y F. Hayes-Roth, *Pattern-Directed Inference Systems*, Academic Press, 1978.

J. AGUSTI-CULLEL
G. VALIENTE
Centre d'Estudis Avançats de Blanes, CSIC