Aplicación de Algebras Fork en la verificación automática de sistemas especificados en Lógica Modal

Lic. Ricardo H. Medel
Área de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto
rhmedel@exa.unrc.edu.ar

Lic. Gabriel A. Baum
Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA)
Facultad de Informática
Universidad Nacional de La Plata
gbaum@info.unlp.edu.ar

Resumen

En este trabajo los sistemas de software son especificados por medio de un grafo y las propiedades que debe cumplir son expresadas como fórmulas de la lógica modal KPI. Ambos, sistema y propiedades, son traducidos a relaciones de un álgebra Fork. Con estas relaciones resultantes se puede alimentar al sistema RELVIEW, el cual permite verificar automáticamente si dichas propiedades se verifican en el sistema diseñado.

1. Introducción

A medida que se expande la aplicación de las computadoras en áreas críticas, los sistemas de software crecen en complejidad, y la puesta a punto y la corrección de errores en los sistemas adquiere una mayor relevancia, llegando a representar aproximadamente el 70% del costo final de los sistemas [Pressman 87] [Zeil 96]. Sin embargo, aún no existen herramientas ni métodos totalmente aceptados por la comunidad informática para realizarlas.

Con el fin de mejorar la calidad de los sistemas producidos, se han desarrollado métodos formales que se aplican en cada paso del proceso de desarrollo a fin de asegurar la correcta construcción del software. El enfoque general es abstraer los sistemas en modelos matemáticos, dar la especificación del sistema deseado en términos de propiedades matemáticas que se deben cumplir y finalmente controlar que los modelos verifican dichas propiedades [Bultan 96]. Estas técnicas de verificación formal revisten especial interés pues permiten demostrar con certeza que el sistema desarrollado cumple con ciertas especificaciones.

Sin embargo, la prueba formal de propiedades de los sistemas es un método complejo, en principio sólo aplicable por expertos entrenados en métodos matemáticos, y que, debido al tiempo que toma aplicarlas, resultan ser económicamente convenientes sólo en grandes proyectos [Atlee 96]. Para que su aplicación en la industria se generalice es necesario que se desarrollen herramientas que permitan el chequeo automático de un sistema, o la traducción automática de un diseño dado de una forma a otra forma más útil [Clements 95], lo cual permitirá disminuir los tiempos y costos de la aplicación de dichos métodos.

Citando a Clarke y Wing [Clarke 96]: "*Progress in the area depends on doing fundamental research, inventing new methods and building new tools, integrating different methods to work togheter, and making concerted efforts by researchers to work with practitioners to transfer technology effectively*"

En particular, para promover la utilización en la práctica de especificaciones formales se debe proveer un lenguaje accesible a ambos, diseñadores y programadores [Wing 87]. Dado que tradicionalmente en Ingeniería de Software se utilizan técnicas de Diseño Estructurado que permiten especificar sistemas de software definiendo un conjunto de módulos y las inter-relaciones entre esos

módulos, el método aquí propuesto utiliza un lenguaje gráfico para expresar el diseño del sistema. Siguiendo la propuesta de Areces e Hirsch [Areces 96], el diseño es representado como un grafo, cuyos vértices son los módulos del sistema y los arcos del grafo son las relaciones entre dichos módulos.

Por otra parte ver al diseño de un sistema de software como un grafo dirigido, tiene una notable similitud con los modelos de las lógicas modales. Esta analogía permite asociar al grafo de diseño del sistema un modelo de Kriepke de la lógica KPI. Así las propiedades del diseño que se desean verificar en dicho sistema se escriben como fórmulas modales.

Luego ambos, sistema y propiedades, son traducidos a un álgebra Fork utilizando la traducción de Frias y Orlowska [Frias 98b]. Con el álgebra de relaciones binarias resultante se alimenta al sistema RELVIEW, producido en la Universidad de Kiel [Behnke 97], el cual permite al desarrollador verificar automáticamente si su sistema cumple con las propiedades deseadas.

De este modo, todo sistema expresable por un grafo puede ser representado en nuestro método y todas las propiedades expresables en una lógica modal pueden ser verificadas automáticamente. La principal fortaleza del método propuesto se basa entonces en la relativa independencia del método elegido para diseñar el sistema y sus requerimientos, permitiendo la verificación automática de las propiedades deseadas del sistema.

Este trabajo está dividido en seis secciones. En la segunda sección se formaliza la noción de especificación de un sistema de software como un grafo dirigido y se explica la lógica modal KPI, para luego establecer el modo de expresar el grafo del sistema como un modelo de esta lógica. En la tercera sección se desarrolla el tema de las álgebras Fork y la traducción del sistema modal a una álgebra Fork. Mientras que en la cuarta sección se utiliza dicha traducción en conjunto con el sistema RELVIEW para la verificación automática de propiedades del sistema desarrollado. La quinta sección expone la posibilidad de aplicar este método al diseño y verificación de sistemas con propiedades temporales. En la sexta sección se establecen las conclusiones y las propuestas de futuros trabajos en el tema. Finalmente, para ilustrar la aplicación de nuestro método en el Anexo bosquejamos un ejemplo muy simple.

2. Especificación de sistemas y sus propiedades

2.1. Diseño del sistema

Un sistema de software puede ser visto como un conjunto de partes (módulos, programas, funciones, variables, etc.) que dependen unas de otras. Las relaciones de dependencia pueden ser: *usa*, *invoca*, *es_parte_de*, *hereda*, etc.

Al utilizar la metodología de Diseño Estructurado, un sistema de software se especifica descomponiendo la solución del problema en módulos y estableciendo gráficamente las inter-relaciones entre los módulos. Un módulo es una parte del diseño que está unívocamente determinada por un nombre y su rol estará definido por las relaciones con otros módulos.

El sistema de software resultante es descripto por medio de un grafo de diseño sobre cuyas relaciones puede realizarse la verificación formal de propiedades. También es posible obtener nuevas relaciones aplicando operaciones, tales como composición, inversa, unión, etc., sobre las relaciones básicas.

Son muchos los trabajos que proponen la visualización gráfica del diseño del sistema como primer paso para luego chequear formalmente sus propiedades, tales como GraphLog [Cosens 92] y GraSp [Agustí 95], mientras que la recientemente desarrollada Arquitectura de Software da impulso a este enfoque, con herramientas como Darwin (de Software Architect's Assistant) [Duval 96].

Generalizando, todo sistema de software puede ser formalizado como un grafo G de la forma:

$$G=<N, E, LN, LE, R_{LN}, R_{LE}>$$

donde:

N: conjunto de nodos o módulos,
$E \subseteq N \times N$: conjunto de arcos entre nodos,
LN: conjunto de rótulos de nodo,

LE: conjunto de rótulos de arco (etiquetas),
R_{LN}: N→LN: función total que asigna a cada nodo un rótulo,
$R_{LE} \subseteq E \times LE$: relación que asigna una etiqueta a varios arcos. Todo arco tiene por lo menos una etiqueta, es decir $\Pi_1(R_{LE})=E$.

Al formalizar así el diseño gráfico de un sistema, se obtiene independencia del formalismo gráfico utilizado, permitiendo que el gráfico sea expresado como la 6-upla que lo define formalmente.

2.2. La lógica modal KPI

Ver al diseño de un sistema de software como un grafo dirigido, tiene una notable similitud con los modelos de las lógicas modales. Esta analogía permite aplicar herramientas modales para el diseño de sistemas, agregando la posibilidad de verificación formal de propiedades del sistema. En este sentido, la propuesta de Areces e Hirsch [Areces 96] asocia al grafo de diseño del sistema un modelo de Kriepke de la lógica KPI. Así las propiedades del diseño que se desean verificar se escriben como fórmulas modales, lo que permite que ambos, modelo y fórmula, sean provistos a un *model-checker* que verifica si dicha fórmula se cumple en el modelo, es decir, en el sistema diseñado.

El nombre KPI indica las características de la lógica: la K indica que se verifica el axioma K ($[](p \rightarrow q) \rightarrow ([]p \rightarrow []q)$), la P que es Polimodal, es decir, tiene un conjunto de relaciones de alcanzabilidad identificadas cada una por un rótulo de un conjunto L, con lo cual las fórmulas de la lógica deben expandirse para considerar los operadores modales [R] y <R> para cada R∈L y, finalmente, la I indica la posibilidad de utilizar los operadores modales inversos $[R]_i$ y $<R>_i$.

Dado un conjunto finito L de rótulos y un conjunto P de símbolos proposicionales, API_L es el alfabeto de la lógica KPI y está formado por $\{\vee, \neg, T, (,)\} \cup \{[a] | a \in L\} \cup \{[a]_i | a \in L\} \cup P$.

Considerando $p_i \in P$, φ y ψ como fórmulas bien formadas de KPI_L y a∈L, entonces las fórmulas bien formadas de KPI_L son:

T,
p_i,
$\neg\varphi$,
$\varphi \wedge \psi$,
$\varphi \vee \psi$,
$<a>\varphi$,
$<a>_i\varphi$.

A partir de estos operadores pueden definirse el resto:

$F \equiv \neg T$,
$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$,
$\varphi \leftrightarrow \psi \equiv \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$,
$[a]\varphi \equiv \neg<a>\neg\varphi$,
$[a]_i\varphi \equiv \neg<a>_i\neg\varphi$.

Un modelo de Kriepke para la lógica KPI_L es la estructura $M=<W, \{a^{\wedge} | a \in L\}, v>$, con W un conjunto no vacío de "mundos posibles", L un conjunto finito de rótulos, cada $a^{\wedge}$ una relación en W×W y v: P×W→{0,1} una valuación modal.

La noción de validez de una fórmula en un mundo $w \in W$ del modelo M se define:

$M,w \models T$,
$M,w \models p_i \Leftrightarrow v(p_i,w)=1$, con $p_i \in P$,
$M,w \models \neg\varphi \Leftrightarrow$ no se verifica $M,w \models \varphi$,
$M,w \models (\varphi \wedge \psi) \Leftrightarrow M,w \models \varphi$ y $M,w \models \psi$,
$M,w \models (\varphi \vee \psi) \Leftrightarrow M,w \models \varphi$ o $M,w \models \psi$,
$M,w \models <a>\varphi \Leftrightarrow \exists w' \in W$, $a^{\wedge}(w,w')$ implica $M,w' \models \varphi$,
$M,w \models <a>_i\varphi \Leftrightarrow \exists w' \in W$, $a^{\wedge}(w',w)$ implica $M,w' \models \varphi$.

Una fórmula φ es *válida* en un modelo M si y sólo si $\forall w \in W$: $M,w \models \varphi$, y se denota como $M \models \varphi$.

La técnica de “model checking” permite chequear si la estructura que modeliza el sistema satisface cierta fórmula, la cual describe alguna propiedad deseada del mismo [Bultan 96]. El problema con esta técnica es la “explosión de estados” que se produce al combinar gran cantidad de componentes secuenciales con espacios de estado finitos en un sistema concurrente [Apt 97].

2.3. Traducción del diseño del sistema a la lógica modal KPI

Para obtener la especificación modal del sistema se asocia a su grafo de diseño G un modelo de Kriepke M=<W, {e^}, *v*> donde:

W={w_i | i ∈ N}: es el conjunto de mundos, para cada nodo i (con rótulo l_i) del grafo hay un mundo w_i.

{e^}={e^ | e∈LE}: es el conjunto de relaciones, una por cada etiqueta del grafo.
Donde e^={(w_k, w_j)∈W×W | ((k,j),e)∈R_{LE}}: el mundo w_j va a ser “e-alcanzable” desde el mundo w_k si en G hay un arco (k,j) con etiqueta e.

v:P×W→{0,1}: la valuación *v* es una función total que asigna a cada variable proposicional (o símbolo de proposición) en cada mundo un valor de verdad {0,1}.

El conjunto P de variables proposicionales debe ser finito. Además, para ciertas variables en particular, *v* estará definida de la siguiente forma:

v(p, w_j) = 1 si y sólo si (j,r) ∈ R_{LN}, es decir R_{LN}(j)=r.

Es decir que hay un símbolo proposicional p por cada rótulo r de módulo y ese símbolo sólo se hará verdadero en el mundo w_i que representa al módulo m_i. Puede pensarse la variable proposicional p como “el nombre del módulo es r”.

Las restantes variables proposicionales (que no representan nombres de módulos) representarán propiedades que se verifican en los distintos mundos, de esa forma la valuación de cada variable proposicional representará el conjunto de mundos donde esa variable vale 1.

La ventaja de expresar el diseño de un sistema de software como un modelo en Lógica Modal KPI_L es que las propiedades que se desean verificar también pueden escribirse en este formalismo. La lógica modal provee al diseñador con una herramienta flexible, que permite expresar propiedades en forma natural y que en muchos casos tiene mejores propiedades computacionales que formalismos más ricos, tales como la lógica de primer orden [Blackburn 94].

3. Traducción del modelo al paradigma relacional

3.1. Verificación de sistemas utilizando métodos relacionales

El álgebra relacional es un formalismo que comparte las virtudes de las lógicas modales, con un mayor poder de expresión. Es una semántica natural para estas lógicas no clásicas, tales como la lógica modal. Esto nos permite crear un marco relacional uniforme en el cual puedan aplicarse y mejorarse las técnicas previamente desarrolladas [Schlingloff 97]. El presente trabajo aplica el denominado *programa* de la lógica algebraica, que es traducir los problemas lógicos en cuestiones algebraicas, para luego usar la maquinaria del algebra universal para resolver el problema algebraicamente y finalmente trasladar la solución otra vez a la lógica original [Blackburn 94].

En este sentido, las álgebras Fork son particularmente útiles pues son finitamente axiomatizables y representables [Frias 97] [Frias 95]. Desarrollos recientes en el área permiten utilizar las álgebras Fork como un cálculo de programas, en el cual se utiliza la lógica de primer orden como lenguaje de especificación y ciertas ecuaciones de las fork-álgebras como programas, permitiendo razonar ecuacionalmente sobre las propiedades de las especificaciones. Haciendo posible, además, establecer estrategias de construcción de programas como fórmulas de primer orden sobre relaciones [Haeberer 97] [Frias 97].

La idea que guía nuestro trabajo es representar las fórmulas de la lógica modal KPI como relaciones en un álgebra de relaciones binarias y aplicar el sistema de demostración relacional para ella. La interpretación relacional de esta lógica se realiza por medio de una traducción por la cual los módulos del sistema (antes vistos como mundos del modelo de Kriepke), las propiedades (antes vistas como fórmulas de la lógica modal) y las relaciones de accesibilidad reciben una representación uniforme como relaciones.

Para realizar la traducción de la lógica KPI al álgebra de relaciones binarias, utilizaremos el trabajo de Frias y Orlowska [Frias 98b], en el cual desarrollaron la Lógica Fork (FL, por sus siglas en inglés) a fin de aplicar una teoría de demostración ecuacional a una gran variedad de lógicas no clásicas. Este formalismo ecuacional permite representar en forma uniforme dichas lógicas y simular formas de razonamiento no clásicas.

Utilizando el método de Frias y Orlowska para representación de fórmulas modales en Fork Álgebras, podemos ver el sistema de software como un conjunto de relaciones binarias concretas y traducir las propiedades que debe verificar el sistema a fórmulas de la Lógica Fork.

3.2. Álgebras Fork

Las álgebras Fork tuvieron origen como un cálculo relacional para la especificación y construcción de programas. Los requerimientos de tal cálculo eran que fuese lo suficientemente expresivo como para expresar una amplia variedad de problemas, que pudiese aplicarse a la transformación de programas, que los conocimientos de lógica y matemática necesarios para aplicarlo fuesen mínimos, que las propiedades de los problemas a resolver se trasladasen de forma natural a los pasos de derivación del programa y que fuese lo suficientemente amplio como para permitir que las estrategias de resolución de problemas también puediesen ser especificadas en dicho cálculo [Frias 97].

La clase de las álgebras Fork propias (PFA) es una extensión de la clase de las álgebras de relaciones binarias con un nuevo operador, denominado *fork* (denotado como $\underline{\nabla}$). Este operador induce una estructuración del dominio, de esta forma las relaciones binarias son relaciones sobre un dominio estructurado $<A,*>$, con (*) una función inyectiva.

Definición 1) Se define la clase de las *PFA como un álgebra con dominos R y U, siendo U un conjunto, R un conjunto de relaciones binarias y E una relación binaria sobre el conjunto U:

$$<R, U, \cup, \cap, \neg, \varnothing, E, |, Id, {}^{T}, \underline{\nabla}, *>$$

donde:

1. $\cup R \subseteq E$,
2. $*: U \times U \rightarrow U$ es una función inyectiva cuando su dominio es restringido a E,
3. Si Id denota la relación identidad sobre U, entonces $\{\varnothing, E, Id\} \subseteq R$,
4. R es cerrado bajo unión ($\cup$), intersección ($\cap$), complemento relativo a E ($\neg$), composición relacional ($|$), conversa (^{T}) y fork ($\underline{\nabla}$), con esta última operación siendo definida como:

$$R \underline{\nabla} S = \{<x, *(y,z)>: xRy \text{ and } xSz\}.$$

Definición 2) La clase de las álgebras Fork propias (PFA) se define como Rd*PFA, donde Rd toma el reducto a las estructuras de la forma: $<R, \cup, \cap, \neg, \varnothing, E, |, Id, {}^{T}, \underline{\nabla}>$.

Al igual que las álgrebras relacionales son una versión abstracta de las álgebras de relaciones binarias, la clase de las álgebras Fork abstractas (AFA) es la contraparte abstracta de la clase PFA.

Definición 3) Un álgebra Fork abstracta es una estructura algebraica:

$$<R, +, \bullet, \neg, 0, 1, ;, 1', {}^{T}, \nabla>$$

donde $<R, +, \bullet, \neg, 0, 1, ;, 1', {}^{T}>$ es un algebra relacional y para todo $r,s,t,q \in R$ se verifica:

(Ax.1) $r \nabla s = (r;(1' \nabla 1)) \bullet (s;(1 \nabla 1'))$

(Ax.2) $(r \nabla s);(t \nabla q)^{T} = (r;t^{T}) \bullet (s;q^{T})$

(Ax.3) $(1' \nabla 1)^{T} \nabla (1 \nabla 1')^{T} \leq 1'$

Se definen $\pi = (1' \nabla 1)^{T}$ y $\rho = (1 \nabla 1')^{T}$ como las funciones que proyectan el primer y segundo componente respectivamente de un par construido con una función biyectiva *.

El siguiente teorema muestra que las álgebras Fork abstractas son representables, la demostración se da en [Frias 97b]:

Teorema) Cada álgebra Fork abstracta es representable, es decir, dada un álgebra Fork abstracta A, existe un álgebra Fork propia B y un isomorfismo h: $A \rightarrow B$.

Como consecuencia de este teorema, la teorías de primer orden de AFA y PFA son las mismas, y una semántica natural puede atribuirse a las fórmulas de primer orden sobre relaciones abstractas en términos de relaciones binarias [Frias 98]. Esta propiedad permite, entre otras cosas, que para describir

la clase de las álgebras Fork abstractas sea necesaria sólo una cantidad finita de axiomas, lo cual permitiría la utilización de demostradores automáticos de teoremas para realizar demostraciones sintácticas.

Dada una $A \in PFA$ con conjunto base U_A, es posible indicar cuáles son los elementos que no representan pares: el rango de una relación R se define como $Ran(R)=(R^T;R)\bullet 1'$, entonces $Ran(\neg(1\nabla 1))$ distingue los objetos del conjunto base que no son pares formados con la función (*). Este término usualmente se denota como $1'_U$ y los elementos se denominan *urelementos*. Además 1_U denota el término $1;1'_U$ y $_U1$ denota $1'_U;1$.

Una relación C se dice *constante* si satisface las siguientes propiedades:

1. $C^T;C \leq 1'$ (C es funcional)
2. $C = 1;C$
3. $C;1 = 1$

Cuando son vistas como relaciones concretas, las relaciones constantes tienen un único elemento en su rango (la constante que está siendo representada) y todos los objetos están relacionados con él.

El dominio de una relación R se define como $Dom(R)= (R;R^T)\bullet 1'$ y una relación R es *ideal derecho* si verifica que $R = R;1$, es decir que cada elemento de su dominio está relacionado con todos los elementos del universo. Este tipo de relaciones permiten representar conjuntos, donde el dominio de la relación son los elementos del conjunto representado [Schmidt 93].

3.3. Lógica Fork

En diversos trabajos se han utilizado formalismos relacionales para algebrizar lógicas no clásicas, tales como lógicas modales y la lógica dinámica [Frias 98b] [Orlowska 97] [Demri 94]. Para ello se desarrollaron formalismos ecuacionales capaces de modelar lógicas no clásicas y simular formas de razonamiento no clásicas. La idea principal es representar las fórmulas de la lógica no clásica como relaciones de un álgebra de relaciones binarias y desarrollar un sistema de pruebas relacional para dicha lógica.

Al ser la clase de las álgebras Fork representable y finitamente axiomatizable, entonces un formalismo basado en esta clase resulta ser apropiado para la formalización relacional de las lógicas no clásicas [Frias 97]. Se desarrolló entonces la lógica Fork (FL) que permite interpretar fórmulas de una lógica no clásica como fórmulas de FL por medio de una traducción que preserva demostrabilidad [Frias 98b]. Bajo esta traducción las fórmulas, antes entendidas como conjuntos de estados, y las relaciones de accesibilidad son ambas representadas uniformemente como relaciones.

A continuación damos una descripción corta de la lógica Fork y sus propiedades, más sobre este formalismo puede encontrarse en [Frias 98b].

Definición 3) El alfabeto de la lógica Fork se define como:

(var) un conjunto numerable RELVAR de *variables relacionales*,
(ls) los *símbolos lógicos*: +, •, ;, ∇, ¬, ^{T}, 1', *0*, 1,
(es) un conjunto numerable de *símbolos extralógicos* (constantes relacionales cuyo significado varía entre modelos),
(as) los símbolos auxiliares "(" y ")".

Definición 4) Una secuencia finita de símbolos del alfabeto de la lógica Fork es una fórmula Fork si y sólo si pertenece a cada conjunto Ω que satisface:

1. RELVAR⊆Ω,
2. los símbolos extralógicos están en Ω,
3. 1', *0* y 1 están en Ω,
4. si R y S están en Ω entonces $R+S$, $R\bullet S$, $R;S$, $R\nabla S$, $\neg R$ y R^T están en Ω.

Definición 5) Una estructura Fork es un par F=(A, *v*) donde A es un álgebra Fork y *v* es una función que asigna relaciones en A a las variables en RELVAR y a los símbolos extralógicos. Por simplicidad, la extensión homomórfica de *v*, *v*':Ω→A será denotada también como *v*.

Definición 6) Una fórmula Fork φ se verifica en una estructura Fork F=(A, *m*), y se denota como $F \models_{FL} \varphi$, si $m(\varphi)=1$ se cumple en A.

Teorema) Toda teoría de primer orden es interpretable en FL, es decir, dada una teoría de primer orden Ψ y una sentencia X, existe un conjunto de fórmulas Fork F_Ψ y una fórmula Fork t_X tal que:

$$\Psi \models X \Leftrightarrow F_\Psi \models_{FL,Ur} t_X.$$

Con *Ur* denotando la ecuación $1;1'_U;1=1$. La demostración de este teorema está dada en [Frias 98b].

3.4. Traducción del sistema

Dados el modelo de Kripke M=<W, {e^}, *v*>, un mundo $w \in W$ y una fórmula modal α, existe una estructura Fork con urelementos:

$$F = (A, v)$$

y un mundo relacional *w*' tal que:

$$M,w \models \alpha \Leftrightarrow F,w' \models_{FL} T'_M(\alpha)$$

donde A es un álgebra Fork libre con generadores W, y T'_M es una traducción de fórmulas modales a fórmulas Fork.

Que el álgebra A tenga como generadores al conjunto W de mundos, significa que por cada mundo modal w_i habrá un átomo a_i perteneciente al universo del álgebra.

La traducción T'_M se define de la siguiente manera:

$T'_M(T) = 1$,
$T'_M(p_i) = P_i$, donde p_i es una variable proposicional y P_i una variable relacional,
$T'_M(\neg\alpha) = 1'_U; \neg(T'_M(\alpha))$,
$T'_M(\alpha\wedge\beta) = T'_M(\alpha)\bullet T'_M(\beta)$,
$T'_M(\alpha\vee\beta) = T'_M(\alpha)+T'_M(\beta)$,
$T'_M(<R>\alpha) = R;T'_M(\alpha)$
$T'_M(<R>_i\alpha) = R^T;T'_M(\alpha)$

La valuación de una variable relacional P_i, que es la traducción de una variable proposicional modal p_i, será una relación ideal derecho que representa el conjunto de mundos modales en los cuales la variable p_i vale 1.

Como se vio en la sección 2.3, hay una variable proposicional p_i por cada módulo del sistema, que indica el nombre del módulo m_i. Entonces la valuación $v(p_i)$ retornará la relación ideal derecho M_i que representa al módulo m_i del sistema modelado. El dominio de dicha relación contendrá sólo el átomo a_i, que representa al único mundo modal w_i donde esa variable p_i es válida.

Adaptando el trabajo de [Frias 98b] a nuestra representación relacional de mundos modales, podemos establecer que dada una relación M_i que representa el mundo modal w_i y una fórmula modal α:

$$F,M_i \models_{FL} T'_M(\alpha) \Leftrightarrow M_i \bullet v(T'_M(\alpha)) \neq 0 \text{ en la Fork Álgebra A.}$$

Es decir, una fórmula modal α se verifica en el mundo relacional M_i si en la Fork Álgebra A la relación $T'_M(\alpha)$ incluye la relación M_i. En particular, dado que ambas relaciones ($T'_M(\alpha)$ y M_i) son ideales derechos, si la fórmula se verifica el resultado de la intersección será la misma relación M_i.

Por lo anterior establecemos que:

$$M,w_i \models \alpha \Leftrightarrow M_i \bullet v'(T'_M(\alpha)) \neq 0 \qquad \text{en A.}$$

Esto es, utilizando el diseño de un sistema, dado por medio un modelo de Kriepke M, podemos construir un modelo relacional A tal que se puede verificar si una propiedad, dada como una fórmula modal α, se cumple en algún módulo m_i del sistema de software representado como el mundo w_i, chequeando si la relación M_i está incluida en la relación resultante de traducir la fórmula α.

Por otro lado, si se quiere verificar que la fórmula α dada se cumple en algún módulo del sistema es suficiente con chequear la validez de la fórmula relacional:

$$v'(T'_M(\alpha)) \neq 0$$

En cambio, si se quiere verificar que la propiedad α se verifica en todo módulo del sistema, la fórmula a validar es:

$$v'(T'_M(\alpha)) = 1.$$

4. Verificación automática de propiedades

4.1. El sistema RELVIEW

Utilizando la técnica de modelización de sistemas y la traducción a una Fork álgebra dadas anteriormente, la verificación de que ciertas propiedades se cumplen en un diseño dado puede realizarse en forma automática utilizando el sistema RELVIEW, producido en la Universidad de Kiel [Behnke 97].

El sistema RELVIEW provee soporte automático, en forma interactiva y visual, para la manipulación de relaciones binarias concretas y descripciones relacionales de dominios [Winter 94]. Usando el ratón, el usuario puede editar relaciones como matrices booleanas o por el grafo que la representa.

Los comandos disponibles, además de permitir ver y manipular las relaciones creadas, permiten aplicar operaciones estándar sobre funciones (unión, intersección, complemento, residuos, cocientes, etc.), realizar chequeos sobre funciones (por vacía, univalente, simétrica, reflexiva, etc.), y también definir funciones y programas aplicables a relaciones.

Algunas de las funciones y operaciones que se pueden usar en RELVIEW para expresar términos relacionales se listan a continuación. Sean `R` y `S` relaciones:

`L(R)`: relación universal con la misma dimensión que `R`,
`0(R)`: relación vacía,
`I(R)`: relación identidad,
`dom(R)`: dominio de `R` como columna,

`-R`: complemento de `R`,
`R|S`: unión de `R` y `S`,
`R&S`: intersección de `R` y `S`,
`R^`: transposición de `R`,
`R*S`: producto entre `R` y `S`,

`trans(R)`: clausura transitiva de `R`,
`refl(R)`: clausura reflexiva de `R`,
`atom(R)`: un átomo (como par) contenido en la relación no vacía `R`,
`empty(R)`: chequea si `R` es vacía.

Durante una sesión de trabajo una colección de relaciones es mantenida en memoria principal, las cuales pueden ser manipuladas y/o referenciadas en los términos relacionales que se ingresen. Las distintas colecciones de relaciones pueden ser almacenadas en archivos para uso futuro.

4.2. Utilización de RELVIEW en la verificación de sistemas

Una vez que el diseño del sistema se traduce a relaciones concretas, cada relación se ingresa a RELVIEW como una matriz Booleana o por medio del grafo que la representa. Luego, se utiliza el sistema RELVIEW para verificar automáticamente si una propiedad dada se cumple en el sistema.

Para modelizar el sistema en RELVIEW por cada módulo m_i del sistema habrá un átomo a_i y se ingresa una relación por cada tipo de conexión entre módulos del sistema (o relación de alcanzabilidad en el modelo modal). De este modo, si los módulos m_k y m_j están conectados por un arco rotulado R en el diseño del sistema, habrá una relación `R` en RELVIEW que contenga al par (a_k,a_j).

Por cada variable proposicional p_i habrá también una relación ideal derecho, que representa el resultado de aplicar la valuación v a la variable p_i. Este tipo de relaciones representan el conjunto de módulos donde la variable p_i es válida. El dominio de esta relación serán los módulos para los cuales p_i vale 1.

Dado que hay una variable proposicional por cada módulo, que es válida sólo en dicho módulo e indica "el nombre del módulo es p_i", habrá una relación ideal derecho M_i por cada módulo m_i del sistema especificado. Esto se ingresa en RELVIEW creando una relación `Mi` tal que el átomo i-ésimo esté relacionado con todos los otros átomos.

Como vimos, para verificar si la fórmula modal α se cumple en un módulo m_i se debe chequear la fórmula relacional $M_i \bullet v(T'_M(\alpha)) \neq 0$. Para realizar automáticamente esta verificación, se debe ingresar en la opción "TEST" la expresión `Mi*`$v(T'_M(\alpha))$, traducida a la notación de RELVIEW. Luego de elegir la casilla de test `EMPTY` y al presionar el botón `TEST`, el sistema indicará si esta expresión relacional es igual a la relación vacía (`0`) o no.

De un modo equivalente se puede verificar si la fórmula α dada se cumple en algún módulo del sistema testeando por `EMPTY` la relación $v(T'_M(\alpha))$, y verificar si la propiedad α se cumple en todo módulo del sistema, ingresando $v(T'_M(\alpha))$ como primera relación y la relación universal (`L`) como segunda relación y luego testeando por `EQUAL`.

También es posible ver exactamente en cuáles módulos del sistema se verifica la propiedad α, ingresando $dom(v(T'_M(\alpha)))$ a la opción "EVAL" la cual retornará el dominio de dicha relación. Los átomos incluidos en el dominio indicarán los módulos donde se verifica la propiedad.

5. Aplicación a sistemas de tiempo real

5.1. Lógica temporal para la especificación de sistemas de tiempo real

Sistemas reactivos con propiedades temporales son denominados sistemas de tiempo real. Muchos formalismos de distinto tipo han sido propuestos para modelar, especificar y verificar propiedades temporales en sistemas de este tipo [Alur 92]. Entre estos formalismos, varias lógicas temporales han surgido en los últimos años [Goldblatt 92] [Manna 95], basadas en la idea original de Pnueli [Pnueli 77] de aplicar la lógica temporal o tensa para la especificación de la conducta de sistemas reactivos.

El término lógica temporal designa a la clase de lógicas modales cuyos operadores modales [] y <> son interpretados como "siempre" y "eventualmente", respectivamente. Esto provee una forma sucinta y natural de expresar propiedades *cualitativas* de sistemas de tiempo real independientes de la velocidad. Por otra parse tiene la limitación de no referirse al tiempo mensurable, por lo que no se pueden especificar propiedades *cuantitativas* de los sistemas. Estas propiedades son conocidas como restricciones de tiempo real "duras" (*hard-real time constraints*) que ponen límites de tiempo al comportamiento de los sistemas reactivos.

A pesar de esta limitación, nuestro método se beneficia de este tipo de lógica temporal simple. La ventaja es que para utilizarla no es necesario ningún cambio de notación, sólo debemos interpretar las relaciones de alcanzabilidad entre vértices del grafo de diseño como relaciones de alcanzabilidad temporal entre estados posibles de un sistema de tiempo real.

5.2. Especificación y verificación de sistemas de tiempo real

Un sistema de tiempo real es un conjunto de secuencias de estados, cada una de las cuales representa un comportamiento posible del sistema. Un sistema de este tipo se define por una 4-upla:

$$RTS = <S, P, \mu, T>$$

donde:

S = conjunto de estados del sistema,

P = conjunto de eventos o proposiciones (sobre estados) observables,

$\mu: S \rightarrow 2^P$, función de rotulado que determina la componente observable de cada estado,

T = conjunto de secuencias de estados, donde si $\tau \in T$ entonces $\tau(i) \in S$ es el único estado del sistema en el instante i.

Dado el grafo de diseño del sistema, tal como lo consideramos en nuestro método, el conjunto de estados S está representado por los vértices del grafo o mundos del modelo de Kriepke del sistema, el conjunto de observables P es el conjunto de variables proposicionales del modelo y la valuación v es la función de rotulado μ. Por lo que pueden ser traducidos siguiendo nuestro método e ingresados como relaciones en RELVIEW.

Las relaciones de alcanzabilidad entre estados, en cambio, pueden estar rotuladas con los eventos que permiten el cambio de un estado a otro, de modo que el conjunto T de secuencias de estados puede obtenerse realizando un ordenamiento topológico (*topological sort*) del grafo de diseño tomando como relación temporal la unión de todas las relaciones rotuladas con eventos.

Habiendo obtenido entonces el RTS como un grupo de relaciones en RELVIEW, las propiedades buscadas del sistema, escritas en lógica temporal, pueden ser traducidas por medio de la función de traducción T'_M e ingresadas en RELVIEW para su verificación automática.

5.3. Verificación de uso de recursos críticos

Una aplicación más de nuestro método, basada en la lógica temporal para la especificación y verificación de sistemas de tiempo real, es la verificación de propiedades de seguridad (*safety*) sobre el acceso a recursos críticos.

Si el grafo de diseño del sistema incluye ciertos vértices rotulados como *recurso crítico*, es decir que puede ser accedido por un único proceso por vez, e incluye también un par de relaciones: una indicando que se accede a un recurso crítico y otra que libera el recurso crítico, se pueden verificar entonces sobre las secuencias de estados ciertas propiedades que debería cumplir el sistema de poseer una buena política de acceso a los recursos críticos.

Veamos algunos ejemplos de tales propiedades. Suponiendo que *Rt* es la relación resultante de aplicar el ordenamiento topológico a las relaciones de alcanzabilidad entre estados del sistema, *Uso* es la relación de acceso desde un estado a un recurso crítico *Rc* y *Libero* la relación que indica que se finaliza el uso de ese recurso crítico *Rc* en cierto estado, algunas de las propiedades que pueden ser verificadas son:

-Que una vez que se accede al recurso crítico, en ningún momento se va a intentar acceder otra vez:

$$<\text{Uso}>\text{Rc} \rightarrow [\text{Rt}]\neg<\text{Uso}>\text{Rc}$$

-Que luego de acceder al recurso crítico, en algún momento en el futuro ese recurso se libera:

$$<\text{Uso}>\text{Rc} \rightarrow [\text{Rt}]<\text{Libero}>\text{Rc}$$

-Que no se libera dos veces el mismo recurso crítico:

$$<\text{Libero}>\text{Rc} \rightarrow [\text{Rt}]\neg(\neg<\text{Uso}>\text{Rc} \wedge [\text{Rt}]<\text{Libero}>\text{Rc})$$

Estas propiedades pueden ser traducidas con T'_M y luego ingresadas en RELVIEW para su verificación automática.

6. Conclusiones y trabajo futuro

Este método de verificación de sistemas presenta varias ventajas interesantes. Por una parte, el diseño gráfico inicial del sistema es de aplicación general, ya que es práctica común en la Ingeniería de Software ver a todo sistema como un conjunto de puntos (módulos, objetos, procesos) relacionados de alguna manera; el formalismo gráfico utilizado está dentro de los conocimientos de los programadores actuales y, dado que permite formalizar distintos tipos de diseños gráficos, puede incluirse en metodologías ya establecidas.

Por otro lado, las lógica modal por medio de la cual se formaliza el grafo y se definen las propiedades a verificarse, es un formalismo ampliamente utilizado para modelar sistemas, incluyendo sistemas de tiempo real. Mientras que la traducción del grafo a un modelo de Kriepke es simple y directa, sin requerir de profundos conocimientos de lógica y matemática.

Finalmente, luego de la traducción del modelo a una Fork álgebra, la verificación de las propiedades es realizada automáticamente por el sistema RELVIEW.

Esta combinación de formalismos (gráfico, lógica modal, álgebra relacional) en un primer momento parecería una desventaja porque exigiría al diseñador conocerlos todos. Sin embargo, esta característica le da mayor poder a la técnica propuesta, dado que las propiedades pueden escribirse por medio de una fórmula modal y luego aplicar la traducción automáticamente, o bien como con una fórmula relacional y trabajar directamente en RELVIEW.

Como se mencionó anteriormente, la principal ventaja del método propuesto se basa en la relativa independencia del formalismo gráfico utilizado para diseñar el sistema y de la lógica modal en la que se expresan sus requerimientos, permitiendo la traducción y verificación automática de las propiedades deseadas del sistema.

Como trabajo futuro queda ver qué pasos, desde el diseño gráfico hasta el modelo relacional ingresado en RELVIEW, pueden automatizarse. La idea general es que el desarrollador de sistemas

ingrese el gráfico de diseño y las propiedades (escritas en Lógica Modal) para que el sistema realice automáticamente todas las traducciones necesarias e informe si se verifican o no tales propiedades. Aunque, según lo mencionado en el párrafo anterior, se debería considerar la posibilidad de ingresar propiedades escritas directamente como fórmulas relacionales.

Otro trabajo futuro en consideración es utilizar un método de diseño en Lógica Modal que exprese en forma abstracta el diseño del sistema por medio de fórmulas modales, las que luego se traducen al formalismo relacional y se ingresan como una teoría en el sistema RALF [Berghammer 94] [Hattensperger 97], para finalmente utilizar este sistema para verificar (semi)automáticamente las propiedades deseadas. Evitando de esta forma las limitaciones surgidas de trabajar con modelos concretos.

Referencias

[Agustí 95] Agustí, J., Robertson, D. and Puigsegur, J., "*GraSp: A GRAphical Specification Language for the Preliminary Specification of Logic Programs*", Institut d'Investigació en Intelligència Artificial (CSIC), Internal Report, 1995.

[Alur 92] Alur, R. and Henzinger, T. A., "*Logics and Models of Real Time: A Survey*", in de Bakker, J.W., Huizing, K., de Roever, W.-P. and Rozenberg, G. (eds.), "*Real Time: Theory in Practice*", LNCS 600, Springer-Verlag, 1992.

[Apt 97] Apt, K.R. and Olderog, E-R., "*Verification of Sequential and Concurrent Programs*", Second edition, Springer, 1997.

[Areces 96] Areces, C.E. e Hirsch, D.F., "*La Lógica Modal como Herramienta de Ingeniería de Software*", Seminario de Tesis, UBA, 1996.

[Atlee 96] Atlee, J., Chechik, M. and Gannon, J., "*Using Model Checking to Analyze Requirements and Designs*", en "*Advances in Computers*", Volume 43, Marv Zelkowitz (editor), Academic Press, 1996.

[Baum 96] Baum, G., Frias, M., Haeberer, A. and Martínez López, P. E., "*From Specifications to Programs: A Fork-algebraic Approach to Bridge the Gap*" en "*Proceedings of MFCS'96*", Cracow, Poland, September 1996, Springer-Verlag, LNCS 1113, págs. 180-191.

[Behnke 97] Behnke, R., Berghammer, R. and Schneider, P., "*Machine Support of Relational Computations: The Kiel RELVIEW* System*", Bericht Nr. 9711, Universität Kiel, 1997.

[Berghammer 94] Berghammer, R. and Hattensperger, C., "*Computer-Aided Manipulation of Relational Expressions and Formulae Using RALF*" in "*Systems for Computer-Aided Specification, Development and Verification*", Bericht Nr. 9416, Universität Kiel, 1994.

[Blackburn 94] Blackburn, P., de Rijke, M. and Venema, Y., "*The Algebra of Modal Logic*", CWI, Amsterdam, Report CS-R9463, 1994.

[Brink 97] Brink, C., Kahl, W. and Schmidt G. (editors), "*Relational Methods in Computer Science*", Springer-Verlag, 1997.

[Bultan 96] Bultan, T., Fischer, J. and Gerber, R., "*Compositional Verification by Model Checking for Counter-Examples*", en [Zeil 96], págs. 224-238.

[Clarke 96] Clarke, E. M. and Wing, J. M., "*Formal Methods: State of the Art and Future Directions*", reporte del Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research, ACM Computing Surveys, December 1996. También en CMU-CS-96-178.

[Clements 95] Clements, P., "*Formal Methods in Describing Architectures*", 1995 Monterey Workshop on Formal Methods and Architecture, available in www.sei.cmu.edu/technology/architecture/projects.html

[Cosens 92] Cosens, M., Mendelzon, A. and Ryman, A., "*Vizualizing and Quering Software Structures*", ICSE'92 Proc. of the 14th Int. Conf. on Software Engineering, 1992.

[Demri 94] Demri, S., Orlowska, E. And Rewitzky, I., "*Towards reasoning about Hoare relations*", in Annals of Mathematics and Artificial Intelligence, 12:265-289, 1994.

[Duval 96] Duval, G. and Cattel, T., "*Specifying and Verifying the Steam-Boiler Problem with SPIN*", in Abrial, J-R., Börger, E. and Langmaack, H. (eds.), "*Formal Methods for Industrial Applications*", LNCS 1165, Springer, 1996, pp. 203-217.

[Frias 95] Frias, M. and Baum, G., "*On the Exact Expressiveness and Provability of Fork Algebras*", en *Abstracts of the Tenth Latinamerican Symposium on Mathematical Logic*, Colombia, 1995.

[Frias 97] Frias, M., Baum, G. and Haeberer, A., "*Fork Algebras in Algebra, Logic and Computer Science*", in *Fundamenta Informaticae*, Vol.32, N°1, págs. 1-25, IOS Press, October 1997.

[Frias 97b] Frias, M., Haeberer, A. M. and Veloso, P. S., "*A Finite Axiomatization for Fork Algebras*", en *Logica Journal of IGPL*, 5(3):311-319, Oxford University Press, 1997.

[Frias 98] Frias, M., Baum, G. and Haeberer, A., "*A Calculus for Program Construction based on Fork Algebras, Generic Algorithms and Design Strategies*", in Orlowska, E. And Szalas, A. (eds.), "*Relationals Methods in Logic, Algebra and Computer Science*", 4th International Seminar RelMiCS, Warsaw, Poland, 14-20 September 1998.

[Frias 98b] Frias, M. and Orlowska, E., "*Equational Reasoning in Non-Classical Logics*", Journal of Applied Non Classical Logic, 8(1-2), 1998.

[Garlan 93] Garlan, D. and Shaw, M., "*An Introduction to Software Architecture*", Advances in Soft. Engineering and Knowledge Engineering, Vol. I, World Scientific Publishing Co., 1993.

[Goldblatt 92] Goldblatt, R. "Logics of Time and Computation", Second Edition, CSLI Lecture Notes No. 7, CSLI Publications, Stanford, 1992.

[Haeberer 97] Haeberer, A., Frías, M., Baum, G. and Veloso, P., "*Fork Algebras*" en (BKS97), págs. 54-69.

[Hattensperger 97] Hattensperger, C., "*RALF: An interactive Proof System for Relationalgebra*", Technical Report, July 31, 1997.

[Loeckx 84] Loeckx, J. and Sieber, K. "*The Foundations of Program Verification*", Wiley-Teubner series in Computer Science, 1984.

[Manna 95] Manna, Z. and Pnueli, A., "*Temporal Verification of Reactive Systems: Safety*", Springer-Verlag, 1995.

[Orlowska 97] Orlowska, E., "*Relational Formalisation of Nonclassical Logics*", in (BKS97), pp. 91-106.

[Pnueli 77] Pnueli, A., "*The temporal logic of programs*", in Proc. 1sth Annual Symp. on Fundations of Computer Science, IEEE Computer Society Press, 1977.

[Pressman 87] Pressman, R. S., "*Software Enginnering: A Practitioner's Approach*", McGraw-Hill, 1987.

[Schlingloff 97] Schlingloff, H. and Heinle, W., "*Relation Algebra and Modal Logics*" in (BKS97), pp. 70-89.

[Schmidt 93] Schmidt, G. and Ströhlein, T., "*Relations and Graphs: Discrete Mathematics for Computer Scientists*", Springer-Verlag, 1993.

[Wing 87] Wing, J., "*Writing Larch Interface Language Specifications*", Transactions on Programming Languages and Systems, 9(1), January 1987.

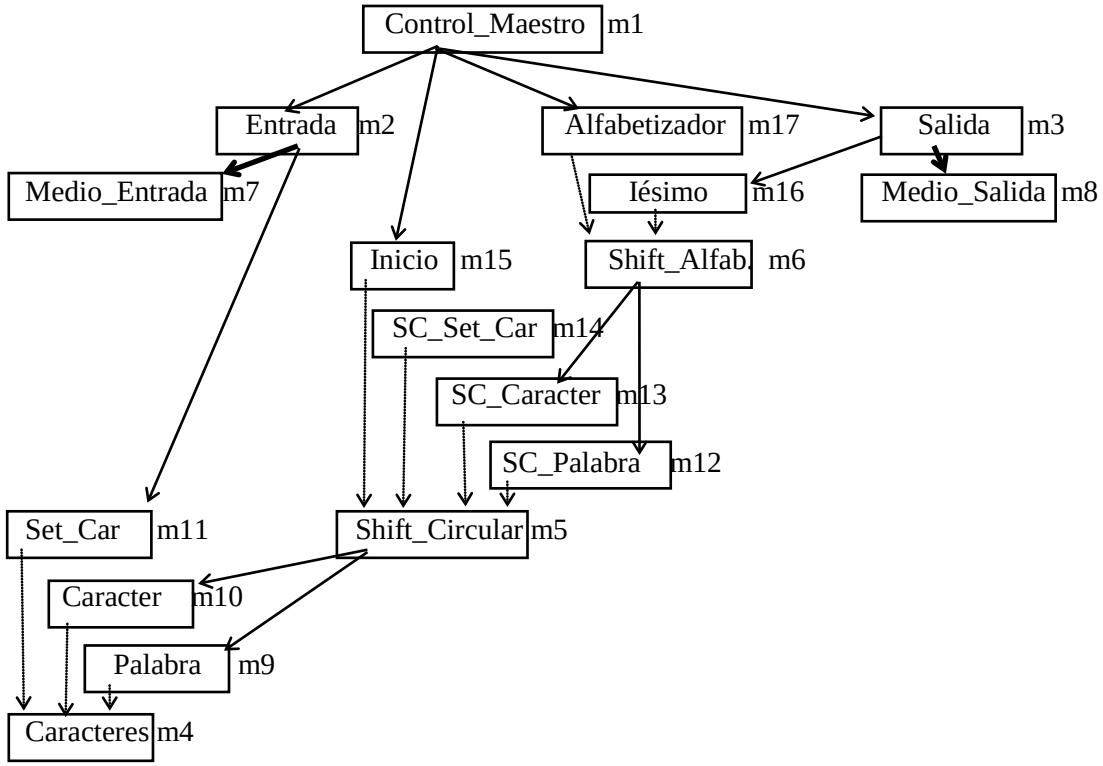
[Winter 94] Winter, M., "*Program Verification With RELVIEW*", in Buth, B. and Berghammer, R. (eds.), "*Systems for computer-aided specification, development and verification*", Bericht N° 9416, Institut für Informatik und Praktische Mathematik, Universität Kiel, 1994.

[Zeil 96] Zeil, S. (editor), "*Proceedings of the 1996 International Symposium on Software Testing and Analysis (ISSTA)*", San Diego CA, January 8-10, 1996.

ANEXO

Ejemplo simple de utilización del método

Para ilustrar la aplicación de nuestro método bosquejamos un ejemplo muy simple basado en una solución para el problema KWIC propuesta en [Areces 96]. Se tiene un sistema cuyos cinco módulos están relacionados entre sí por tres relaciones ("invoca", "es_parte_de" y "entrada_salida"), y se desea verificar si se cumple la propiedad de que "todo TAD utiliza servicios sólo de otros TADs".



Referencias:

→ invoca

⇢ parte {es parte de}

➔ e/s {entrada/salida}

Grafo del sistema:

$$G=<N, E, LN, LE, R_{LN}, R_{LE}>$$

con:

N={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17},

E={(1,2), (1,3), (1,15), (1,17), (2,7), (2,11), (3,8), (3,16), (5,9), (5,10), (6,12), (6,13), (9,4), (10,4), (11,4), (12,5), (13,5), (14,5), (15,5), (16,6), (17,6)}

LN={Control_Maestro, Entrada, Salida, Caracteres, Shift_Circular, Shift_Alfab., Medio_Entrada, Medio_Salida, Palabra, Caracter, Set_Car, SC_Palabra, SC_Caracter, SC_Set_Car, Inicio, I-ésimo, Alfabetizador},

LE={invoca, parte, e/s},

R_{LN}={(1,Control_Maestro), (2,Entrada), (3,Salida), (4,Caracteres), (5,Shift_Circular), (6,Shift_Alfab.), (7,Medio_Entrada), (8,Medio_Salida), (9,Palabra), (10,Caracter), (11,Set_Car), (12,SC_Palabra), (13,SC_Caracter), (14,SC_Set_Car), (15,Inicio), (16,I-ésimo), (17,Alfabetizador)},

R_{LE}={((1,2),invoca), ((1,3),invoca), ((1,15),invoca), ((1,17),invoca), ((2,7),e/s), ((2,11),invoca), ((3,8),e/s), ((3,16),invoca), ((5,9),invoca), ((5,10),invoca), ((6,12),invoca), ((6,13),invoca), ((9,4),parte), ((10,4),parte), ((11,4),parte), ((12,5),parte), ((13,5),parte), ((14,5),parte), ((15,5),parte), ((16,6),parte), ((17,6),parte)}

Modelo de Kriepke del sistema:

$$M=<W, \{invoca, parte, e/s\}, V>$$

con:

W={m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17}

invoca={(m1,m2), (m1,m3), (m1,m15), (m1,m17), (m2,m11), (m3,m16), (m5,m9), (m5,m10), (m6,m12), (m6,m13)}

parte={(m9,m4), (m10,m4), (m11,m4), (m12,m5), (m13,m5), (m14,m5), (m15,m5), (m16,m6), (m17,m6)}

e/s={(m2,m7), (m3,m8)}

V(Control_Maestro,m1)=1,	V(Entrada,m2)=1,	V(Salida,m3)=1,
V(Caracteres,m4)=1,	V(Shift_Circular,m5)=1,	V(Shift_Alfab.,m6)=1,
V(Medio_Entrada,m7)=1,	V(Medio_Salida,m8)=1,	V(Palabra,m9)=1,
V(Caracter,m10)=1,	V(Set_Car,m11)=1,	V(SC_Palabra,m12)=1,
V(SC_Caracter,m13)=1,	V(SC_Set_Car,m14)=1,	V(Inicio,m15)=1,
V(I-ésimo,m16)=1,	V(Alfabetizador,m17)=1,	V(pi,mi)=0 ∀otro (pi,mi).

Relaciones en RELVIEW:

Se debe ingresar en RELVIEW una relación `R` por cada relación en el modelo M y una relación ideal derecho `Mj` por cada módulo m_j ($1 \le j \le 17$).

`M1`={(1,1),(1,2,(1,3),(1,4),(1,5),(1,6),(1,7),(1,8),(1,9),(1,10),(1,11),(1,12),(1,13), (1,14),(1,15),(1,16),(1,17)}

:

`M17`={(17,1),(17,2,(17,3),(17,4),(17,5),(17,6),(17,7),(17,8),(17,9),(17,10),(17,11), (17,12),(17,13),(17,14),(17,15),(17,16),(17,17)}

`invoca`={(1,2), (1,3), (1,15), (1,17), (2,11), (3,16), (5,9), (5,10), (6,12), (6,13)}

`parte`={(9,4), (10,4), (11,4), (12,5), (13,5), (14,5), (15,5), (16,6), (17,6)}

`e_s`={(2,7), (3,8)}

Verificación de la propiedad:

Un Tipo Abstracto de Datos (TAD) es un módulo que define funciones, es decir que un módulo es parte de un TAD si es parte de otro módulo, o sea que vale en él la fórmula modal $<parte>_i T$. Un módulo utiliza servicios de un TAD si todo módulo por él invocado es un TAD, entonces verifica $[invoca]<parte>T$. Entonces la propiedad buscada se denota como:

$$<parte>_i T \rightarrow [invoca]<parte>T.$$

Por medio de la traducción T'_M establecemos

$$T'_M(<parte>_i T \rightarrow [invoca]<parte>T) = T'_M(\neg<parte>_i T \vee \neg<invoca>\neg<parte>T)$$
$$= 1'_U;\neg(parte^T;1) + 1'_U;\neg(invoca;1'_U;\neg(parte;1))$$

Por lo que la propiedad se verifica en el sistema si esta relación es igual a la relación universal 1. Entonces en RELVIEW se verifica, por medio de la opción `TEST`, que la relación establecida por la fórmula:

```
I(invoca)*-(parte^*L(invoca))|
          I(invoca)*-(invoca*I(invoca)*-(parte*L(invoca))
```

sea igual a la relación universal `L(invoca)`, si el resultado es positivo significa que la propiedad buscada se verifica en todo el sistema.