Implementación de un model checker para lógicas modales en Haskell

Marcelo A. Cardós

LIFIA, Departamento de Informática, Universidad Nacional de La Plata.
C.C.11, Correo Central, 1900, La Plata, Buenos Aires, República Argentina.
E-mail: mac@info.unlp.edu.ar
URL: http://www-lifia.info.unlp.edu.ar/

Resumen

Las lógicas modales son utilizadas en una gran variedad de campos relacionados con las ciencias de la computación. Entre ellos se encuentran la verificación de programas, la concurrencia, la teoría de autómatas, los operadores de punto fijo y los sistemas de comunicación distribuídos.

El objetivo de este trabajo es describir la implementación en Haskell de un verificador de modelos genérico para lógicas modales y analizar su comportamiento, ventajas y desventajas con respecto a posibles implementaciones basadas en lenguajes imperativos.

Palabras clave: progamación funcional, lógica modal, model checking

Implementación de un model checker para lógicas modales en Haskell

1 Introducción

Las lógicas modales son utilizadas en una gran variedad de campos relacionados con las ciencias de la computación. Entre ellos se encuentran la verificación de programas, la concurrencia, la teoría de autómatas, los operadores de punto fijo, la lógica dinámica y los sistemas de comunicación distribuídos. Revisiones y descripciones de éstas y otras aplicaciones pueden consultarse en [9, 8, 2, 1, 5] entre otros.

Para verificar programas, se simbolizan los sistemas como fórmulas de algún sistema lógico y luego trata de determinarse la existencia de algún modelo para dicha fórmula. En rigor, se proponen posibles modelos y se verifica su correspondencia con la fórmula. Este procedimiento es el que se conoce con el nombre de verificación de modelos o *model checking* (ver [4]).

Por lo general, estos verificadores representan las fórmulas modales mediante sistemas de transición de estados.

El objetivo de este trabajo es describir la implementación en Haskell [6] de un verificador de modelos genérico para lógicas modales y analizar su comportamiento, ventajas y desventajas con respecto a posibles implementaciones basadas en lenguajes imperativos. Además, se estudiará la facilidad para derivar verificadores específicos para varias lógicas modales tradicionales.

Para ello, a continuación se describen brevemente algunas formulaciones de las lógicas modales y sus modelos estándar (los sistemas de transición antes mencionados). Luego se comenta y analiza la implementación, que puede verse en el apéndice A. Finalmente, se comentarán algunas posibilidades de trabajos futuros y ampliaciones, en particular las referidas a las lógicas polimodales cuyo conjunto de operadores modales tiene alguna estructura algebraica particular.

2 Lógicas modales

2.1 El lenguaje modal

El lenguaje modal proposicional es una extensión del lenguaje proposicional puro, formado mediante el agregado de conectivos unarios (informalmente conocidos como conectivos *box*). Originalmente había un solo conectivo $\Box$, aunque para varios propósitos se vio la necesidad de agregar varios de tales conectivos $[i]$, uno para cada posible elemento i de un conjunto índice I. De esta manera, hay en realidad muchos lenguajes modales posibles, uno por cada posible conjunto índice I. La sintaxis, semántica y teoría de pruebas asociados con los lenguajes modales están diseñados de modo que la lógica proposicional clásica es un caso particular de lógica modal, el que tiene $I = \emptyset$.

Los elementos i de I se llaman *etiquetas* o *labels*, y el propio I se conoce como *signatura* del lenguaje modal. Así, dos lenguajes modales son idénticos si y sólo si tienen la misma

signatura.

A diferencia de los conectivos proposicionales $\neg$, $\longrightarrow$, $\vee$, $\wedge$, $\top$(constante *verdad*) y $\bot$(constante *falsedad*), los conectivos *box* no tienen una interpretación fija. Para cada fórmula ϕ del lenguaje modal, podemos usar $[i]$ para obtener una nueva fórmula $[i]\phi$, que puede ser leída de diversas maneras, y cada lectura sugiere una semántica y un sistema de prueba diferente.

En el caso de los primeros trabajos en lógicas modales, que tenían un solo conectivo modal, la fórmula compuesta $\Box\phi$ podía leerse como "ϕ es necesaria", o "ϕ es obligatoria". Para comprender esta interpretación, hay que pensar en la noción de "mundo posible", de modo que las lecturas anteriores sugieren que ϕ debe ser verdadera en todos los mundos posibles para que $\Box\phi$ sea verdadera. A partir de estas lecturas, en ocasiones se dice que el conectivo *box* expresa la "necesidad" de una fórmula. Ahora bien, para expresar el hecho de que una fórmula pueda ser cierta en algunos mundos y no en otros, habría que escribir una fórmula como $\neg\Box\neg\phi$. Así aparece un nuevo conectivo modal llamado *diamante* que, en esta interpretación, expresa la "posibilidad" de la fórmula ϕ, y se representa como $\Diamond\phi$. Resulta claro que de haber varios operadores *box* (llamado *polimodal*) es posible definir un operador diamante por cada uno de ellos.

Por razones de simplicidad en la descripción, en el resto del trabajo se utilizará solamente el lenguaje monomodal (una sola pareja (*box*, diamante) de conectivos modales).

A continuación se define formalmente el lenguaje utilizado y el conjunto de fórmulas asociado.

Definición 2.1 El lenguaje monomodal queda definido por:

- los elementos P_i de un conjunto fijo numerable Var de *variables*;
- los *conectivos proposicionales* $\top$, $\bot$, $\neg$, $\longrightarrow$, $\longleftrightarrow$, $\wedge$ y $\vee$, de aridades 0, 0, 1, 2, 2, 2 y 2 respectivamente;
- el *conectivo modal* $\Box$, de aridad 1 y
- los *símbolos de puntuación* (y).

Definición 2.2 Las fórmulas del lenguaje se obtienen únicamente por aplicación de las siguientes cláusulas:

atómicas Cada variable $P_i \in Var$ y cada constante $\top$ y $\bot$ es una fórmula.

prop Si θ, ψ, ϕ son fórmulas, entonces $(\neg\phi)$, $(\theta \longrightarrow \psi)$, $(\theta \vee \psi)$ y $(\theta \wedge \psi)$ son fórmulas.

modal Si ϕ es una fórmula, también lo es $\Box\phi$.

Llamaremos $Form$ al conjunto de todas las fórmulas bien formadas.

Adicionalmente utilizaremos el diamante a manera de abreviatura, dada por la ecuación $\Diamond\phi \equiv \neg\Box\neg\phi$.

2.2 Semántica

Hay diversos tipos de modelos para las lógicas modales. Los modelos más primitivos se caracterizan por el conjunto de mundos posibles y una asignación de valores de verdad a las sentencias atómicas en cada mundo posible. Desde este punto de vista, una sentencia A es verdadera en un mundo posible sólo en el caso en que A sea verdadera en todos los mundos posibles.

Debido a que esta definición es muy estricta para su uso práctico, se ha generalizado la noción de modelo introduciendo una relación entre mundos posibles. De este modo, puede ocurrir que A sea verdadera en un mundo posible sin ser A verdadera en todos ellos.

Los modelos estándar incorporan una relación entre los mundos posibles. Así, estos modelos pueden pensarse como una terna $< W, \triangleright, P >$, donde W es el conjunto de los mundos posibles, P define la asignación de valores de verdad a las sentencias atómicas en cada mundo posible y $\triangleright$es la relación.

Formalmente:

Definición 2.3 Un modelo estándar es una estructura $\mathcal{M} =< W, \triangleright, P >$ tal que:

1. W es un conjunto,
2. $\triangleright \subseteq W^2$ y
3. $P : Var \rightarrow 2^W$

La relación $\triangleright$ cambiará significativamente de interpretación de un modelo a otro, pero en general puede pensarse como de accesibilidad, posibilidad relativa o relevancia. Así, anotaremos $\alpha \triangleright \beta$ al decir que el mundo β es accesible desde (o relevante para o posible en relación con) el mundo α. Cabe destacar que $\triangleright$ es una relación cualquiera; no se hace suposición alguna sobre sus propiedades debido a que asignándole diferentes propiedades se obtienen diferentes clases de lógicas modales.

Anotaremos $\models_\alpha^{\mathcal{M}} A$ para decir que A es *verdadera en un mundo posible* α *de un modelo* $\mathcal{M}$. Esta noción se define como sigue:

Definición 2.4 Sean α mundos posibles en el modelo $\mathcal{M} =< W, \triangleright, P >$:

1. $\models_\alpha^{\mathcal{M}} P_i$ sii $\alpha \in P(i)$, para $i \in \mathbf{N}$.
2. $\models_\alpha^{\mathcal{M}} \top$.
3. No es cierto $\models_\alpha^{\mathcal{M}} \bot$.
4. $\models_\alpha^{\mathcal{M}} \neg A$ sii no es cierto $\models_\alpha^{\mathcal{M}} A$.
5. $\models_\alpha^{\mathcal{M}} A \wedge B$ sii son ciertos a la vez $\models_\alpha^{\mathcal{M}} A$ y $\models_\alpha^{\mathcal{M}} B$.
6. $\models_\alpha^{\mathcal{M}} A \vee B$ sii es cierto alguno de $\models_\alpha^{\mathcal{M}} A$ o $\models_\alpha^{\mathcal{M}} B$ o ambos.
7. $\models_\alpha^{\mathcal{M}} A \rightarrow B$ sii cada vez que $\models_\alpha^{\mathcal{M}} A$, también $\models_\alpha^{\mathcal{M}} B$.

8. $\models^{\mathcal{M}}_{\alpha} A \leftrightarrow B$ sii son ciertos a la vez $\models^{\mathcal{M}}_{\alpha} A \rightarrow B$ y $\models^{\mathcal{M}}_{\alpha} B \rightarrow A$.

9. $\models^{\mathcal{M}}_{\alpha} \quad A$ sii para todo $\beta \in \mathcal{M}$ tal que $\alpha \triangleright \beta$, $\models^{\mathcal{M}}_{\beta} A$.

10. $\models^{\mathcal{M}}_{\alpha} \quad A$ sii para algún $\beta \in \mathcal{M}$ tal que $\alpha \triangleright \beta$, $\models^{\mathcal{M}}_{\beta} A$.

Definición 2.5 Se dice que una sentencia A *es verdadera en el modelo* $\mathcal{M}$, y se anota $\models^{\mathcal{M}} A$, sii A es verdadera en todos los mundos posibles de $\mathcal{M}$.

Si A es verdadera en $\mathcal{M}$, se dice que $\mathcal{M}$ *es un modelo para* A. Se dice que $\mathcal{M}$ es modelo de un conjunto de fórmulas Γ sii $\mathcal{M}$ es modelo de todas las fórmulas de Γ.

Otra noción que es muy útil para dar semántica a las fórmulas modales, y que se utilizará en la implementación, es la de *conjunto de verdad* de una fórmula. Intuitivamente, el conjunto de verdad $\|A\|^{\mathcal{M}}$ de una fórmula A en un modelo $\mathcal{M}$ es el conjunto de todos los mundos posibles de $\mathcal{M}$ en los que A es verdadera. Formalmente:

Definición 2.6 $\|A\|^{\mathcal{M}} = \{\alpha \in \mathcal{M} / \models^{\mathcal{M}}_{\alpha} A\}$

Para el caso de las fórmulas atómicas y proposicionales, puede verse que

Teorema 2.7 *Si* $\mathcal{M} = <W, \triangleright, P>$ *es un modelo, entonces:*

1. $\|P_i\|^{\mathcal{M}} = P(i)$, *para cada* i = 0,1, ...
2. $\|\top\|^{\mathcal{M}} = W$
3. $\|\bot\|^{\mathcal{M}} = \emptyset$
4. $\|\neg A\|^{\mathcal{M}} = W - \|A\|^{\mathcal{M}}$
5. $\|A \wedge B\|^{\mathcal{M}} = \|A\|^{\mathcal{M}} \cap \|B\|^{\mathcal{M}}$
6. $\|A \vee B\|^{\mathcal{M}} = \|A\|^{\mathcal{M}} \cup \|B\|^{\mathcal{M}}$
7. $\|A \longrightarrow B\|^{\mathcal{M}} = W - \|A\|^{\mathcal{M}} \cup \|B\|^{\mathcal{M}}$
8. $\|A \longleftrightarrow B\|^{\mathcal{M}} = W - \|A\|^{\mathcal{M}} \cup \|B\|^{\mathcal{M}} \cap W - \|B\|^{\mathcal{M}} \cup \|A\|^{\mathcal{M}}$

También puede verse que:

Corolario 2.8 *Sea* $\mathcal{M} = <W, \triangleright, P>$ *un modelo*

1. $\|A \longleftrightarrow B\|^{\mathcal{M}} = W - \|A\|^{\mathcal{M}} \triangle \|B\|^{\mathcal{M}}$
2. $\|A\|^{\mathcal{M}} \subseteq \|B\|^{\mathcal{M}}$ *sii* $\models^{\mathcal{M}} A \longrightarrow B$
3. $\|A\|^{\mathcal{M}} = \|B\|^{\mathcal{M}}$ *sii* $\models^{\mathcal{M}} A \longleftrightarrow B$

La relación de los modelos estándar puede ser reemplazada equivalentemente por una función que asocia a cada mundo posible del modelo con un conjunto de mundos posibles.

Dado un modelo estándar $\mathcal{M} =< W, \triangleright, P >$, se define la función

$f : W \to 2^W$

$f(\alpha) = \{\beta \in \mathcal{M} / \alpha \triangleright \beta\}$.

Esto significa que $f(\alpha)$ es el conjunto de los mundos en $\mathcal{M}$ que están $\triangleright$-relacionados con α; intuitivamente, $f(\alpha)$ es el conjunto de los mundos accesibles desde α.

Si se utiliza f en lugar de $\triangleright$, la estructura $< W, f, P >$ resultante es también un modelo estándar, y las condiciones de verdad para A y A dadas en 2.4 pueden reemplazarse por:

$$\models^{\mathcal{M}}_{\alpha} \quad A \text{ sii para todo } \beta \in f(\alpha), \models^{\mathcal{M}}_{\beta} A \tag{2.1}$$

$$\models^{\mathcal{M}}_{\alpha} \quad A \text{ sii para algún } \beta \in f(\alpha), \models^{\mathcal{M}}_{\beta} A \tag{2.2}$$

Las condiciones de verdad anteriores pueden relacionarse con los conjuntos de verdad de la siguiente manera:

Corolario 2.9 *Sea* $\mathcal{M} = < W, f, P >$ *un modelo*

1. $\models^{\mathcal{M}}_{\alpha} \quad A$ *sii* $f(\alpha) \subseteq \|A\|^{\mathcal{M}}$
2. $\models^{\mathcal{M}}_{\alpha} \quad A$ *sii* $f(\alpha) \cap \|A\|^{\mathcal{M}} \neq \emptyset$

3 La implementación

3.1 Descripción

Como se dijo, una fórmula A es verdadera en un modelo $\mathcal{M} =< W, \triangleright, P >$ si es verdadera en todos los mundos posibles del modelo. Esto es fácilmente verificable mediante la comprobación de que el conjunto de verdad de A, $\|A\|^{\mathcal{M}}$, coincide con el conjunto W de mundos posibles del modelo, lo que permite que el verificador de modelos se defina mediante la siguiente función:

```
mc          :: Model s → MForm → Bool
mc  m  f    =  truthSet m f == states m
```

donde `truthSet` es la función que calcula el conjunto de verdad asociado a una fórmula en un modelo, y `states` calcula el conjunto de los mundos posibles del modelo. (Aclaración: el nombre `states` proviene de pensar al grafo de la relación de accesibilidad como un diagrama de transición de estados, por lo que en lo sucesivo se utilizarán indistintamente los nombres mundo posible y estado).

Resulta claro, a partir de la definición anterior, que la idea fundamental de la implementación reside en el cálculo del conjunto de verdad de la fórmula dada. A fin de ver cómo se lleva a cabo dicho cálculo, se describen a continuación los elementos involucrados.

Como se vio antes, los modelos estándar son ternas que constan de un conjunto de estados, una relación entre estados y una función de verdad que indica para cada variable

en cuáles estados se considera verdadera. Esto da lugar a que el tipo de los modelos pueda definirse como sigue:

$$\textbf{type}\ \textsc{Model}\ \mathit{state} = (\textsc{Set}\ \mathit{state}, \textsc{Relation}\ \mathit{state}, \textsc{Prop} \to \textsc{Set}\ \mathit{state})$$

Para que los modelos estén correctamente definidos, deben especificarse con claridad el conjunto de mundos posibles y el de variables. Acerca del conjunto de variables, se sabe que es numerable y que sus elementos se anotan como P_i, con $i \in \mathbf{N}$, de modo que la función de verdad puede pensarse como una sucesión de conjuntos de estados en los que es verdadera la variable correspondiente al índice de la sucesión. De allí que se identifique el tipo de las proposiciones atómicas PROP con el tipo INT. De este modo sólo resta conocer el tipo de las entidades que serán consideradas estados del sistema, el cual, dado que no hay restricciones previsibles, se recibe como parámetro.

La implementación propuesta se compone, hasta ahora, de tres módulos Haskell que responden a respectivas categorías conceptuales: el módulo `ModelChecker` define al verificador de modelos ya descripto; el módulo `ModalLogic` describe todos los conceptos básicos de la lógica modal (el tipo de los modelos y sus accesorios, el tipo de las fórmulas modales, el predicado de satisfacción de las fórmulas, la función `truthSet`); y el módulo `Set` contiene el tipo abstracto de los conjuntos.

Si bien en esta implementación el tipo de los modelos no es abstracto, está previsto que lo sea en una futura versión que incluirá una herramienta visual para la construcción de los modelos y las fórmulas. Sin embargo, se lo ha utilizado como tal en gran medida. Es así como se dispone de funciones de proyección sobre la terna de los modelos que tienen nombres específicos, como la antes mencionada `states`. De este modo el código resulta ser más legible y autoexplicativo.

A partir del concepto de conjunto de verdad (def. 2.6), surge naturalmente la siguiente definición de la función `truthSet`:

```
truthSet          ::  Model s → MForm → Set s
truthSet  m  p    =   subset (satisfy m p) (states m)
```

donde `satisfy` es la función que verifica si una fórmula es verdadera en un mundo posible del modelo y `subset` p s calcula el subconjunto de s que verifica un cierto predicado p.

El teorema 2.7 muestra que para las fórmulas atómicas los conjuntos de verdad pueden calcularse directamente, mientras que para las fórmulas proposicionales puede hacerse mediante operaciones entre los conjuntos de verdad de las subfórmulas intervinientes. Así, puede darse para estos casos una definición recursiva en términos de sí misma de la función `truthSet`, la cual puede codificarse utilizando el mecanismo de coincidencia de patrones o *pattern matching*. Para los casos en que la fórmula está afectada por un operador modal, se utiliza la definición original, debido a que no hay una operación de conjuntos capaz de describir la situación que sea independiente del problema en cuestión. Además, la definición de satisfacción de una fórmula en un mundo posible permite determinar la verdad o no de la fórmula a partir de la verdad de sus componentes y el corolario 2.9, da un método para la búsqueda de los estados que hacen verdadera una fórmula modal.

En virtud de que este método utiliza los conjuntos de verdad, las definiciones obtenidas para las funciones `truthSet` y `satisfy` son mutuamente recursivas. La única dificultad que podría representar este hecho es que se ralentizaría el cálculo si hubiera involucrados muchos operadores modales en la fórmula.

En virtud de la decisión de utilizar las propiedades del corolario 2.9, se hace necesario destacar que la relación del modelo se implementará, para simplificar los cálculos, mediante la función asociada que calcula el conjunto de los estados accesibles desde uno dado, en lugar trabajar con la visión tradicional como conjunto de pares ordenados.

Finalmente, los conjuntos se manipulan como un tipo abstracto, y se representan mediante el siguiente tipo algebraico:

$$\textbf{data}\ \textsc{Set}\ a = \textsc{Set}\ [a]\ a \to \textsc{Bool}$$

donde en $\textsc{Set}\ x_s\ p$, x_s contiene los elementos del conjunto y p representa el predicado de pertenencia.

La idea básica de la representación pasa por la frase "los conjuntos pueden representarse por su predicado de pertenenecia", pero se le hizo el agregado de la lista debido a que hay operaciones (como `subset` o `isEmpty`) en las que es necesario conocer a todos los elementos. Estos detalles son relevantes para el análisis del comportamiento global del sistema y su complejidad.

3.2 Análisis

El objetivo ha sido desarrollar un verificador de modelos que funcione para cualquier lógica modal. Hasta el momento han resultado infructuosas las búsquedas de una herramienta similar. Sólo se han encontrado algunas más específicas, restringidas a una lógica fija. Sin embargo, resulta apropiado analizar algunos aspectos que tienen que ver con la dificultad y complejidad algorítmica de un desarrollo de estas características.

Con el análisis realizado hasta ahora podría hacerse una implementación en cualquier lenguaje. Las diferencias relevantes pasan por la elección de las estructuras de datos, y para este caso la decisión más importante pasa por la determinación de la representación de la relación de accesibilidad. Pensando en un lenguaje imperativo, puede decirse que cualquiera que ella sea, en los casos más críticos del proceso (es decir, aquéllos en los que hay involucrados operadores modales) se está obligado a recorrer exhaustivamente el grafo de una u otra manera a fin de establecer su verdad. Por su parte, la versión funcional que se ofrece en este artículo, si bien trabajando sobre conjuntos, termina teniendo exactamente la misma dificultad en los mismos casos. Es interesante observar, por ejemplo, que la estructura de datos propuesta para representar el tipo abstracto de los conjuntos coincide con la definición de un grafo mediante listas de adyacencias. De este modo, puede concluirse que cualquier implementación de una herramienta de estas características tendrá una complejidad algorítmica teórica similar.

Con todo, la implementación propuesta aquí tiene la ventaja de la claridad descriptiva y puede aprovechar la existencia de herramientas formales para la derivación de programas como medio de encontrar verificadores de modelos más específicos.

Por su parte, las implementaciones imperativas tendrían la ventaja de poder ajustar algunos detalles que mejoren la eficiencia de los algoritmos, especialmente en términos de dependencias de las arquitecturas subyacentes. Está claro que, siendo así, el esfuerzo de desarrollo es mucho mayor. Además, no resulta muy claro que las modificaciones requeridas para obtener verificadores más específicos puedan hacerse sobre la base de una implementación genrérica previa.

4 Conclusiones y trabajos futuros

Esta implementación constituye una descripción concisa y clara de un verificador de modelos que funciona sobre una lógica modal cualquiera. Esto está garantizado por el método mismo de su construcción. Por la misma razón, no puede asegurarse que sea posible obtener, en cualquier paradigma, una herramienta con una complejidad teórica menor a la observada.

Resulta claro que la primera extensión en que se puede pensar para este desarrollo es darle una aspecto más útil mediante el diseño de las herramientas visuales mencionadas antes. Además, parece natural intentar derivar implementaciones de verificadores más específicos, que respondan a restricciones en el conjunto de axiomas de la lógica utilizada o a determinación de propiedades de la relación de accesibilidad; por supuesto, aprovechando herramientas formales de transformación de programas.

Por otro lado, también es interesante explorar la expansión producida al incorporar más operadores modales. Esto origina que los modelos pasen a depender de conjuntos de relaciones y por ende se amplía el campo de investigación cuando uno se pregunta si el hecho de que el conjunto de relaciones tenga una cierta estructura algebraica puede influir en el desarrollo de los verificadores asociados y en qué medida.

A Código fuente

```
module ModalLogic where
import Set
data MForm  =  Atom Int          |  Top
            |  Bottom            |  Not MForm
            |  Box MForm         |  Diamond MForm
            |  And MForm MForm   |  Or MForm MForm
            |  Cond MForm MForm  |  Bicond MForm MForm
--
-- Función de cálculo de los conjuntos de verdad
--
```

```
truthSet                 ::  Model s -> MForm -> Set s
truthSet m (Atom p)      =   truth m p
truthSet m (Top)         =   states m
truthSet m (Bottom)      =   empty
truthSet m (Not p)       =   states m `minus` truthSet m p
truthSet m (Box p)       =   let p1 = satisfy m (Box p)
                             in subset p1 (states m)
truthSet m (Diamond p)   =   let p1 = satisfy m (Diamond p)
                             in subset p1 (states m)
truthSet m (And p q)     =   let s1 = truthSet m p
                             in let s2 = truthSet m q
                              in s1 `intersection` s2
truthSet m (Or p q)      =   let s1 = truthSet m p
                             in let s2 = truthSet m q
                              in s1 `union` s2
truthSet m (Cond p q)    =   let s1 = states m `minus` truthSet m p
                             in let s2 = truthSet m q
                              in s1 `union` s2
truthSet m (Bicond p q)  =   let s1 = truthSet m p
                             in let s2 = truthSet m q
                              in states m `minus` (s1 `minus` s2)
--
-- Un modelo es una terna que consta de un conjunto
-- de estados, una relación entre los estados y
-- una función de verdad.
type Model state  =  (Set state, Relation state, Truth state)
type Relation s   =  s -> Set s
type Truth s      =  Int -> Set s
--
-- Funciones de proyección del modelo
--
states              ::  Model s -> Set s
states (w, r, p)    =   w
relation            ::  Model s -> Relation s
relation (w, r, p)  =   r
truth               ::  Model s -> Truth s
truth (w, r, p)     =   p
```

```
satisfy                    ::  Model s -> MForm p -> s -> Bool
satisfy m (Atom p) s       =   s `belongs` (truth m p)
satisfy m (Top) s          =   True
satisfy m (Bottom) s       =   False
satisfy m (Not p) s        =   not (satisfy m p s)
satisfy m (Box p) s        =   let r = relation m s
                               in let s = truthSet m p
                                in r `included` s
satisfy m (Diamond p) s    =   let r = relation m s
                               in let s = truthSet m p
                                in let s' = r `intersection` s
                                in (not .  isEmpty) s'
satisfy m (And p q) s      =   let p1 = satisfy m p s
                               in let p2 = satisfy m q s
                                in p1 && p2
satisfy m (Or p q) s       =   let p1 = satisfy m p s
                               in let p2 = satisfy m q s
                                in p1 || p2
satisfy m (Cond p q) s     =   let p1 = satisfy m p s
                               in let p2 = satisfy m q s
                                in (not p1) || p2
satisfy m (Bicond p q) s   =   let p1 = satisfy m p s)
                               in let p2 = satisfy m q s
                                in (p1 && p2) || not (p1 || p2)
module Set  (Set, empty, isEmpty,fromList, intersection,
            union, minus, subset, image, belongs, included)
where
data Set a  =  Set [a] (a -> Bool)
empty  ::  Set a
empty  =   Set [] (\x -> False)
isEmpty              ::  Set a -> Bool
isEmpty (Set xs p)   =   null xs
addElem                ::  a -> Set a -> Set a
addElem x (Set xs p)   =   Set (x:xs)
                           (\v -> (v == x)||(p v))
belongs                ::  a -> Set a -> Bool
belongs x (Set xs p)   =   p x
union                            ::  Set a -> Set a -> Set a
union (Set xs p1) (Set ys p2)    =   Set (g p2 xs ys)
                                     (\v -> (p1 v)||(p2 v))
intersection                       ::  Set a -> Set a -> Set a
intersection (Set xs p1) (Set ys p2)   =   Set (filter p2 xs)
                                           (\v -> (p1 v)&&(p2 v))
```

```
minus                 ::  Set a -> Set a -> Set a
minus s1 (Set ys p)   =   s1 `intersection` (Set ys (not.p))
subset                 ::  (a -> Bool) -> Set a -> Set a
subset p (Set xs p1)   =   (Set xs p1) `intersection` (Set xs p)
fromList  ::  [a] -> Set a
fromList   =  foldr addElem empty .  nodups
image                 ::  (a -> b) -> Set a -> Set b
image f (Set xs p1)   =   fromList (map f xs)
included                            ::  Set a -> Set a -> Bool
included (Set xs p1) (Set ys p2)   =   h p2 xs
-- FUNCIONES AUXILIARES

-- remueve elementos duplicados de una lista
nodups     ::  [a] -> [a]
nodups xs   =  concat [x:filter (/=x) xs | x <- xs]
-- g p xs ys = filter (not.p) xs ++ ys
g p    []    ys  =  ys
g p  (x:xs)  ys  =  let
                     new = g p xs ys
                    in if (p x)
                     then new
                     else (x:new)
-- h p xs = and (map p xs)
h p []      =  True
h p (x:xs)  =  (p x) && (h p xs)
module ModelChecker where
import ModalLogic

-- El verificador requiere un modelo y una fórmula a
-- verificar.
mc      ::  Model s -> MForm -> Bool
mc m f   =   truthStates m (normalize f) == states m
```

Referencias

[1] S. Abramsky and S. Vickers. Quantales, observation logic and process semantics. En *Mathematical Structures in Computer Science*, pp. 161–227, 1993.

[2] B. Banieqbal and H. Barringer. Temporal logic with fixed points. En *Temporal Logic in Specification*, volumen 398 de *Lecture Notes in Computer Science*, pp. 1–20. Springer Verlag, 1989.

[3] B. F. Chellas. *Modal Logic: an introduction*. Cambridge University Press, 1980.

[4] E. A. Emerson. Temporal and modal logic. En *Handbook of Theoretical Computer Science*, volumen B, pp. 995–1072. Elsevier, 1990.

[5] D. Harel. Dynamic logic. En *Handbook of Philosophical Logic*, volumen II, pp. 497–604. D. Reidel and Co., 1984.

[6] J. Peterson, K. Hammond, et al. *Report on the Programming Language Haskell, A Non–Strict, Purely Functional Language. Version 1.3.* Yale University, Mayo 1996.

[7] S. Popkorn. *First Steps in Modal Logic.* Cambridge University Press, 1994.

[8] C. Stirling. Comparing linear and branching time temporal logics. En *Temporal Logic in Specification*, volumen 398 de *Lecture Notes in Computer Science*, pp. 1–20. Springer Verlag, 1989.

[9] C. Stirling. Temporal logics for CCS. En *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volumen 350 de *Lecture Notes in Computer Science*, pp. 660–672. Springer Verlag, 1989.